Заведующему кафедрой
фундаментальной информатики
А. Г. Смольянову
студента 4 курса
очной формы обучения
(на (бес)платной основе)
направления подготовки
02.03.02 – Фундаментальная информатика
и информационные технологии
факультета математики и
информационных технологий
ФГБОУ ВО «МГУ им. Н.П. Огарёва»
Уварова Глеба Сергеевича

заявление

Прошу разместить мою выпускную квалификационную работу на тему «Разработка web-сервиса потокового вещания на основе программной платформы NODE.JS» в электронной библиотечной системе университета в полном объеме.

09.06.2020 г.
Дата

Подпись

Заявление о самостоятельном характере выполнения работы

Я, Уваров Глеб Сергеевич, обучающийся 4 курса направления подготовки 02.03.02 Фундаментальная информатика и информационные технологии, заявляю, что в моей работе на тему «РАЗРАБОТКА WEB-СЕРВИСА ПОТОКОВОГО ВЕЩАНИЯ НА ОСНОВЕ ПРОГРАММНОЙ ПЛАТФОРМЫ NODE.JS», представленной в Государственную экзаменационную комиссию для публичной защиты, не содержится элементов неправомерных заимствований.

Все прямые заимствования из печатных и электронных источников, а также ранее защищенных письменных работ, кандидатских и докторских диссертаций имеют соответствующие ссылки.

Я ознакомлен с действующим в Университете Положением о проверке работ студентов, обучающихся в ФГБОУ ВО «МГУ им. Н.П. Огарёва», на наличие заимствований, в соответствии с которым обнаружение неправомерных заимствований является основанием для отрицательного отзыва руководителя работы.

09.06.2020 г.

Подпись обучающегося — Дата

Работа представлена для проверки в Системе «Антиплагиат.ВУЗ»

09.06.2020 г.

Дата представления работы — *подпись руководителя*

ЛИСТ ОЗНАКОМЛЕНИЯ

(защита ВКР)

Я, Уваров Глеб Сергеевич, студент 4 курса очной формы обучения направления подготовки 02.03.02 Фундаментальная информатика и информационные технологии (профиль «Информатика и компьютерные науки») факультета математики и информационных технологий , заявляю, что ознакомлен с пунктом 1.4 Положения о проведении государственной итоговой аттестации по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры в ФГБОУ ВО «МГУ им. Н. П. Огарёва»:

пункт 1.4. Обучающимся и лицам, привлекаемым к ГИА, во время ее проведения запрещается иметь при себе и использовать средства связи.

09.06 2020 г.

дата

подпись студента

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Мордовский государственный
университет имени Н. П. Огарева

СПРАВКА

о результатах проверки текстового документа на наличие заимствований

Проверка выполнена в системе Антиплагиат.ВУЗ

Автор работы	**УВАРОВ ГЛЕБ СЕРГЕЕВИЧ**
Факультет, кафедра, номер группы	ФМиИТ, ФИ, 402
Тип работы	Выпускная квалификационная работа
Название работы	Разработка web-сервиса потокового вещания на основе программной платформы Node
Название файла	ВКР - Анти - УваровГС - ФИИТБ) - 2020.docx
Процент заимствования	**10,87%**
Процент цитирования	**0,35%**
Процент оригинальности	**88,78%**
Дата проверки	**16:47:54 09 июня 2020г.**
Модули поиска	Кольцо Вузов; Модуль поиска общеупотребительных выражений; Коллекция Патенты; Модуль поиска перефразирований Интернет; Модуль поиска перефразирований eLIBRARY.RU; Модуль поиска "МГУ им. Н. П. Огарева"; Коллекция Медицина; Модуль поиска Интернет; Коллекция Гарант; Коллекция eLIBRARY.RU; Модуль поиска переводных заимствований по интернет (EnRu); Модуль поиска переводных заимствований по elibrary (EnRu); Переводные заимствования; Цитирования; Коллекция РГБ; Сводная коллекция ЭБС; Модуль выделения библиографических записей; Модуль поиска ИПС "Адилет"
Работу проверил	СМОЛЬЯНОВ АНДРЕЙ ГРИГОРЬЕВИЧ ФИО проверяющего
Дата подписи	09.06.2020г.
	Подпись проверяющего

Чтобы убедиться в подлинности справки, используйте QR-код, который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.

О Т З Ы В

на выпускную квалификационную работу студента 4 курса
направления подготовки 02.03.02 Фундаментальная информатика и информационные технологии
Уварова Глеба Сергеевича
на тему «Разработка web-сервиса потокового вещания на основе программной платформы Node»

Настоящая выпускная квалификационная работа посвящена интернет-технологиям, а именно, вопросам разработки программного обеспечения для организации вещания в прямом эфире посредством интернета. Подобные возможности позволяют существенно изменить многие сферы общественной жизни, что определяет тему настоящего исследования как весьма актуальную.

Структурно работа состоит из введения, трёх глав и заключения.

Задачей автора настоящей бакалаврской работы было изучение теории кодирования видеопотока данных. Целью работы было создание web-сервиса, который позволил бы транслировать видеоданные в интернет в прямом эфире. Проделанное исследование потребовало от автора изучения ряда важных вопросов, необходимых для практической реализации программного обеспечения, а именно, 1) изучить понятие кодека и способы сжатия видеоизображения; 2) разобраться с общим механизмом трансляции видео в интернет; 3) изучить протоколы видеопередачи в интернет; 4) разработать собственный web-сервис потокового вещания на основе изученного теоретического материала.

Представленная работа сделана грамотно, творчески. Автор достаточно глубоко изучил теоретические вопросы, необходимые для реализации практической части работы. Текст работы хорошо оформлен, структурирован, иллюстрирован многочисленными графическими материалами.

Изложенный в работе материал имеет несомненную практическую значимость с точки зрения теории и практики программирования. Проделанное автором исследование характеризует его как программиста высокой квалификации, владеющего парадигмами, концепциями и методами современного программирования и способного к реализации своих знаний в реальных проектах, связанных с разработкой программного обеспечения.

Считаю, что настоящая работа удовлетворяет требованиям, предъявляемым к выпускным квалификационным работам, а её автор, Уваров Г. С., заслуживает оценки «отлично».

Научный руководитель
зав. кафедрой фундаментальной информатики,
доцент А. Г. Смольянов

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н. П. ОГАРЁВА»**

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ
Зав. кафедрой
Канд. физ.-мат. наук, доц.
А. Г. Смольянов
(подпись)
«18» июня 2020 г.

БАКАЛАВРСКАЯ РАБОТА

РАЗРАБОТКА WEB-СЕРВИСА ПОТОКОВОГО ВЕЩАНИЯ НА ОСНОВЕ ПРОГРАММНОЙ ПЛАТФОРМЫ NODE.JS

Автор бакалаврской работы (подпись) 09.06.2020 (дата) Г. С. Уваров

Обозначение бакалаврской работы БР–02069964–02.03.02–18–20

Направление 02.03.02 Фундаментальная информатика и информационные технологии

Руководитель работы
канд. экон. наук, доц. (подпись) 09.06.2020 (дата) А. Г. Смольянов

Нормоконтролер
канд. экон. наук, доц. (подпись) 11.06.2020 (дата) С. В. Гарина

Саранск
2020

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н. П. ОГАРЁВА»**

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ
Зав. кафедрой
Канд. физ.-мат. наук, доц.
_____ А. Г. Смольянов
(подпись)
«25» декабря 2019 г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
(в форме бакалаврской работы)

Студент Уваров Глеб Сергеевич

1 Тема «Разработка web-сервиса потокового вещания на основе программной платформы NODE.JS»

Утверждена приказом № 10037-с от 25.12.2019

2 Срок представления работы (проекта) к защите _____

3 Исходные данные для научного исследования:

4 Содержание бакалаврской работы:

4.1 Тезисный обзор технологии потокового вещания.

4.2 Основные принципы работы онлайн трансляций

4.3 Разработка и проектирование WEB-сервиса потокового вещания

5 Приложения: код программы

Руководитель работы (проекта) _____ 25.12.2019
подпись, дата

Задание принял к исполнению _____ 25.12.2019
подпись, дата

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н. П. ОГАРЁВА»**

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ
Зав. кафедрой
Канд. физ.-мат. наук, доц.
_____ А. Г. Смольянов
(подпись)
«18» июня 2020 г.

БАКАЛАВРСКАЯ РАБОТА

РАЗРАБОТКА WEB-СЕРВИСА ПОТОКОВОГО ВЕЩАНИЯ НА ОСНОВЕ ПРОГРАММНОЙ ПЛАТФОРМЫ NODE.JS

Автор бакалаврской работы Г. С. Уваров
(подпись) (дата)

Обозначение бакалаврской работы БР–02069964–02.03.02–18–20

Направление 02.03.02 Фундаментальная информатика и информационные технологии

Руководитель работы
канд. экон. наук, доц. А. Г. Смольянов
(подпись) (дата)

Нормоконтролер
канд. экон. наук, доц. С. В. Гарина
(подпись) (дата)

Саранск
2020

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н. П. ОГАРЁВА»**

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ
Зав. кафедрой
Канд. физ.-мат. наук, доц.
_____ А. Г. Смольянов
(подпись)
«25» декабря 2019 г.

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
(в форме бакалаврской работы)**

Студент Г. С. Уваров

1 Тема «Разработка web-сервиса потокового вещания на основе программной платформы NODE.JS»

Утверждена приказом № 10037-с от 25.12.2019

2 Срок представления работы (проекта) к защите _____

3 Исходные данные для научного исследования:

4 Содержание бакалаврской работы:

4.1 Тезисный обзор технологии потокового вещания.

4.2 Основные принципы работы онлайн трансляций

4.3 Разработка и проектирование WEB-сервиса потокового вещания

5 Приложения: код программы

Руководитель работы (проекта) _____
подпись, дата

Задание принял к исполнению _____
подпись, дата

РЕФЕРАТ

Бакалаврская работа содержит страницы, 34 рисунка, 16 использованных источников, 3 приложения.

NODE.JS, HTML, CSS, WEB-СЕРВИС, СТРИМИНГ, ИНТЕРНЕТ, ПОТОКОВОЕ ВЕЩАНИЕ.

Объектом исследования является процесс разработки web-сервиса потокового вещания с рассмотрением алгоритмов сжатия и их передачи по протоколу в интернет.

Цель работы – рассмотрение технологических особенностей создания web-сервиса с использованием механизмов передачи трансляции.

В результате проведенной работы были изучены технологии, используемые в настоящее время для разработки стриминговых web-сайтов, проанализированы особенности и возможности алгоритмов сжатия видеопотока, а также способы передачи этих видеоданных в интернет. Так же мною бы разработан стриминговый web-сервис с использованием этих технологий.

Степень внедрения – частичная.

Область применения – сеть интернет.

Эффективность – высокая безопасность и скорость загрузки web-сайта, очень низкая стоимость поддержки сайта.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодня интернет стал одной из самых прогрессивных технологий. Одна из этих технологий позволила применять вещание в прямом эфире в интернете. Если раньше вещание было только по телевизору и было доступно не всем людям, то сегодня с развитием интернета транслировать в прямом эфире может каждый человек.

Широкое распространение интернета позволяет абсолютно каждому пользователю создавать свою трансляцию. Телеканалы и радиостанции теперь не обходятся только телевизионным форматом. Теперь они так же организовывают на своих сайтах трансляции в интернете.

Это даёт увеличенную сферу влияния как для пользователя, так и для крупных компаний. Проводить вещания для узких круг лиц, проводить маркетинговые исследования, а также напрямую взаимодействовать с пользователями и давать быструю обратную связь.

Такая организация позволяет транслировать в любую точку планеты посредством глобальной сети. Вещание может осуществляться и по прямому эфиру, так и в записи. Помимо этих возможностей, так же можно анализировать различную статистику просмотров, прослушиваний и многое другое.

Онлайн-трансляции на сегодняшний день стали очень важны и популярна среди интернет-пользователей, поскольку согласно данным, телевизор уже на порядок отстает от интернета и люди уже больше предпочитают смотреть новости и развлекательный контент на таких платформах, как YouTube, Twitch. Так же за последнее время очень сильно вырос показатель онлайн трансляций, где каждый человек мог бы вещать свою трансляцию и представлять для аудитории исключительно специальный контент, который не представляет нам телевизор. Данное положение и определила актуальность нашей бакалаврской работы.

Целью данной бакалаврской работы является изучение теории кодирования видеопередачи данных и разработка с помощью этих возможностей web-сервиса, который позволит транслировать видеоданные в интернет в прямом эфире. В соответствии с целью, были сформулированы следующие задачи:

– Ознакомиться с кодеком, способах сжатия видеоизображения;

– Ознакомиться с общим механизмом трансляции в интернет;

– Ознакомиться с протоколами видеопередачи в интернет;

– Разработать собственный web-сервис потокового вещания на основе изученного материала.

Бакалаврская работа состоит из введения, трёх глав и заключения.

В первой главе описывается актуальность использования этой технологии и сфера применения.

Во второй главе рассматривается теоретическая информация о кодировании и декодировании видео, способах сжатии, и его применение, а также о способах передачи видеоданных в интернет, какие существуют протоколы, их преимущества и недостатки.

В третьей главе с помощью рассмотренных технологий, спроектирован и разработан web-сервис, который будет транслировать видеоизображение в интернет.

1. Тезисный обзор технологии потокового вещания

1.1 Определение потокового вещания

Потоковое вещание – это доставка мультимедийных потоков информации (видео, аудио) удаленным пользователям, в реальном режиме времени [1].

Так же можно встретить следующие обозначения термина, и они являются равнозначными:

- Живая трансляция
- Потоковое видео
- Стриминг
- Прямая трансляция
- Онлайн трансляция

Это общий термин, который используется, как и в эфире телепередач, так и в интернете. Мы предполагаем, что в данной работе эти термины относятся в рамках web-сервиса, то есть вещание будет происходить в интернете. Поскольку технология вещания для телевизионных каналов происходит немного другим образом.

Стриминговый сервис – это равнозначный термин к web-сервису потокового вещания.

1.2 Актуальность технологии

Развитие технологий сжатия видео и аудио сигналов открыло широкие возможности передачи вещательной информации в масштабе реального времени не только по выделенным каналам связи, но и через сеть интернет. Потоковое вещание в интернет (streaming) завоевало популярность несмотря на то, что качество звука и изображения пока еще не достигло уровня blu-ray качества. Зато оперативность доставки и неисчерпаемость вещательных ресурсов привлекли множество энтузиастов.

Многим людям технология вещания в интернете представляется очень простой. Действительно, все технологические процессы кодирования исходных сигналов и ретрансляция их для группы пользователей могут быть организованы в простейшем виде на базе распространенных персональных компьютеров. И только профессионалы понимают, что качество, надёжность и массовость вещания в интернет даются дорогой ценой и требуют научного изучения [2].

Очень неожиданный скачок этой технологии придал внезапно неожиданный всплеск пандемии коронавируса COVID-19. В результате введения карантина по всему миру, значительное число профессий перешло на удалённую работу, что позволило увеличить приток различных трансляций по интернету.

В России все учащиеся и студенты перешли на удалённое обучение из дома и для образовательных учреждений выпал довольно уникальный новый случай - испытать новую систему обучения и преподавания, а именно обучение из дома с помощью трансляций и конференций. Так как подобный случай происходит в эпоху информационной эры, то актуальность данной технологии и её внедрении очень важна.

Как и любая технология, эта не стоит на месте и стремительно развивается. Возможно, через несколько лет потоковая передача видео откроет перед пользователями возможности, которые казались фантастикой еще вчера.

1.3 Сфера применения потокового вещания в интернете

Сегодня сфера применения этой технологии относительно обширна и применяется в различных сферах жизни: от удалённой работы в конференции до массовых вещаний в интернете. Мы рассмотрим несколько главных источников, где используется потоковое вещание.

1 **Просмотр видео в интернете**. Как известно, в Глобальной Паутине найдется все – главное уметь искать. Пользователи могут посмотреть любой фильм, мультфильм или видеоклип в удобное время.

2 **Видеосвязь**. Популярные мессенджеры, поддерживающие эту функцию – например, Skype, используют те же технологии и протоколы. Кроме того, видеосвязь сегодня способно обеспечить большинство популярный социальных сетей, а также многие специализированные ресурсы. Стоит отметить, что видеосвязь может не только удовлетворить потребность в общении и развлечении, но находит и практическое применение во время организации обучающих вебинаров или онлайн-конференций.

3 **Просмотр трансляции с web-камер.** Не покидая собственной квартиры, любой пользователь может посмотреть на известные достопримечательности, прочие интересные места или на нашу планету с борта МКС. Однако развлечением использование web-камер не ограничивается: в последнее время все большую актуальность обретает наблюдение за избирательными участками во время выборов, что помогает обнаружить нарушения. Широко применяется видеонаблюдение службами безопасности, однако в этом случае трансляция ведется не через интернет, а по локальной сети.

4 **Стриминг**. Благодаря доступности технологий при наличии соответствующего оборудования любому пользователю интернета доступно создание собственной трансляции. Более того, для тысяч энтузиастов это стало основным источником дохода: зритель при желании может поддержать стримера, пожертвовав ему определенную сумму. В этой сфере лидерство стабильно остается за киберспортсменами, ведущими трансляцию игрового процесса [3].

Таким образом, мы рассмотрели основные сферы, где используется потоковое вещание. Наш web-сервис будет предполагать именно сферу стриминга с упором на количественную аудиторию и высокого качества трансляции с определённой задержкой, вместо трансляций реального времени.

2 Основные принципы работы онлайн трансляций

2.1 Способы передачи потокового видео в интернете

Для просмотра потокового видео через интернет на устройстве пользователя достаточно иметь лишь браузер (Google Chrome, Firefox, Yandex Browser, Opera).

Для трансляции потокового мультимедиа обычно используется один из двух способов:

– **Последовательный** – видеофайл воспроизводится с жесткого диска компьютера пользователя или сервера провайдера услуг. Как правило, при передаче таким способом качество изображения и звука выше. К недостаткам стоит отнести то, что невозможно переключить ролик с одного момента на другой, не дождавшись его буферизации. То есть, интересующий фрагмент должен быть загружен, чтобы пользователь смог его просмотреть. Для трансляции подходит обычный web-сервер.

– **В реальном времени** – требует наличия потокового сервера. Этот метод больше подходит для передачи видеофайлов большой длительности. Пользователь может выбрать место, с которого он хочет начать просмотр. Также этот вид потокового мультимедиа вещания используется для трансляции с web-камеры или захвата экрана.

2.2 Общий механизм работы онлайн трансляций

Организация онлайн-трансляции состоит из двух основных этапов:

1 Сжатие видеоизображения;

2 Передача сжатого видеоизображения по широковещательному протоколу;

Разберем первый пункт. Сначала определим вообще понятие изображения, выясним из чего это изображение состоит. Мы все понимаем, что наш экран состоит из пикселей. В наших мониторах обычно принят

стандарт **RGB** (Read-Green-Blue). Этот стандарт описывает красный, зеленый и синий как комбинацию цветов. Каждый из этих цветов имеет диапазон цвета от 0 до 255, что соответствует 8 битам в двоичной системе счисления. Интенсивность цвета требует определенного хранения – этот параметр называется **битовая глубина**. Так как модель RGB у нас имеет три плоскости, то на каждую цветовую плоскость приходится 8 бит, тогда наша глубина цвета будет равна 8 бит * 3 плоскости = 24 бита = 3 байта. Более наглядное представление на рисунке 1.

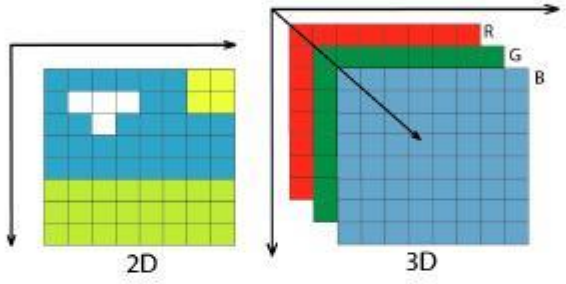

Рисунок 1 – принцип отображения цвета.

Мы разобрали принцип работы отображения цветов. Теперь разберем из чего состоит видеоизображение. Мы все понимаем, что обычное изображение состоит из 1 кадра, то есть это просто картинка. Видеоизображение представляет собой последовательность картинок, состоящие из N кадров за некий период времени, где N – количество кадров в секунду.

Количество бит в секунду, которое требуется для передачи видеоизображения, называется **скоростью передачи** или **битрейтом**.

Сам битрейт вычисляется по следующему принципу:

*битрейт = ширина * высота * кадров в секунду * битовая глубина*

Например, возьмем обычный ролик длиной 5 секунд с размерами экрана 1920x1080 пикселей и частотой кадров 30 в секунду. Тогда видео будет весить немного немало порядка

*(1920 * 1080) байт * 30 кадров * 3 байта * 5 секунд = 933 120 000 байт = 933Мб*

Технологий, которые бы передавали такие данные в интернет с высокой скоростью, для того чтобы смотреть трансляции пока не изобретены. Каналы передачи и возможности хранения ограничены. Именно по этой причине, нам необходимо применить сжатие видео с минимально возможным размером и максимально хорошим качеством изображения.

Во втором пункте мы затрагиваем информацию о том, как передавать наши сжатые видеоданные в прямой поток. Для этого существуют два основных транспортных протокола TCP и UDP. Однако используются не они, а их надстройки над этими протоколами.

Например, RTMP протокол является надстройкой над TCP, а такие как MPEG и HLS над UDP. Есть определённая разность между ними в сложности реализации. Но сегодня, для массового вещания не используют протоколы, использующие надстройки над TCP по той причине, что TCP гарантирует доставку и последовательность пакетов. Это означает то, что если произойдёт потеря доставки при видеопередаче, то это означает, что пропускная способность у пользователя резко сократится, будут возникать задержки и смотреть трансляцию будет очень некомфортно. У нас остаётся единственный вариант, использовать протокол UDP.

Его преимущество заключается в том, что если по пути у нас потеряется какая-то определённая часть пакета информации, то скорее всего пользователь не почувствует особого дискомфорта, так, как например потеря 1 кадра не сыграет роли и пользователь попросту не заметит. При этом трансляция будет идти беспрерывно и без задержек. Именно поэтому мы будем использовать протоколы, основанные на UDP.

2.3 Алгоритм сжатия видеоизображения

Мы выяснили, что основной проблемой при работе с видеоданными, является колоссальное использование дискового пространства. Хранение даже короткого видео потребует огромных пространственных ресурсов.

В результате на сегодняшний день используются алгоритмы сжатия с потерями, такие чтобы соответствовали максимальному уменьшению размера и при этом сохраняя качество видеопотока [4].

При сжатии используются следующие избыточности:

– **пространственная** – используется избыточность в хранении памяти.

– **временная** – используется межкадровая избыточность, так как разница между кадрами в большинстве случаев за единицу времени – минимальная.

– **цветовая** – Цветовая избыточность. Из-за специфики человеческого глаза человек воспринимает больше яркость картинки, нежели цветность [5].

Разберем с конца. Цветовая избыточность заключается в том, что из-за специфики человеческого глаза человек воспринимает яркость, нежели цветность изображения. В этом можно хорошо убедиться, в соответствии с картинкой (Рисунок 2).

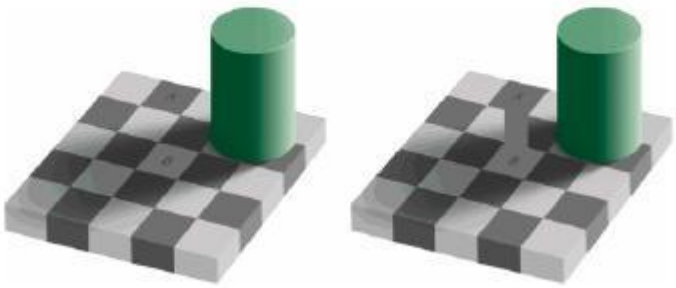

Рисунок 2 – Иллюзия цветности

Несмотря на якобы разный цвет в квадратах A и B левой картинке, на самом деле они одинакового цвета. В правой картинке, если добавить перемычку, то это сразу станет понятно, что цвета абсолютно одинаковые.

Поэтому можно использовать альтернативы цветового пространства из RGB, например в YCbCr, а сам процесс сжатия цветности называют цветовой субдискретизацией.

Цветовая субдискретизация — технология кодирования изображений со снижением цветового разрешения, при которой частота выборки цветоразностных сигналов может быть меньше частоты выборки яркостного сигнала [6]. Она уменьшает требуемую пропускную способность канала передачи (а значит, и максимальный битрейт) без существенного влияния на качество изображения.

Самой распространенной цветовой моделью является **YbCbCr**, где **Y** – представление яркости, **Cb** – насыщенный синий, **Cr** – насыщенный красный. Используя эту модель, можно создавать аналогичные цветные изображения, как видим ниже(Рисунок 3):

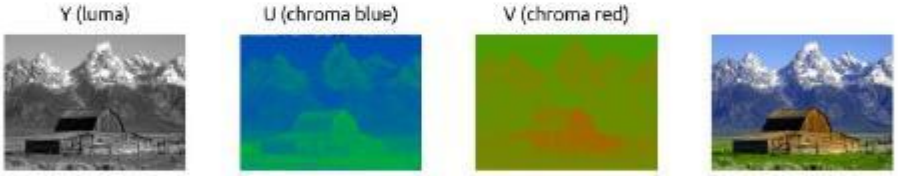

Рисунок 3 – различные компоненты слоёв

Чтобы получить такое изображение, используется стандарт **BT.601**, в котором используется следующая формула преобразования:

$$\mathbf{Y} = 0.299\mathbf{R} + 0.587\mathbf{G} + 0.114\mathbf{B}$$

После получения яркости изображения можно вывести синий и красный цвет:

Cr = 0.564*(**B - Y**)

Cb = 0.713*(**R - Y**)

Работает так и обратное преобразование в систему **RGB**:

R = **Y** + 1.402**Cr**

B = **Y** + 1.772**Cb**

G = **Y** - 0.344**Cb** - 0.714**Cr**

Структура дискретизации сигнала обозначается соотношением **X:a:b**.

X – эталон горизонтальной выборки (как правило, равен 4)

a – количество выборок цветности в первой строке пикселей (горизонтальное разрешение по отношению к a)

b – количество изменений выборок цветности между первой и второй строками пикселей.

Наиболее используемые соотношения в современных кодеках используются следующие схемы:

– **4:4:4** – без субдискретизации, то есть RGB

– **4:2:2** - Используется в научных исследованиях, профессиональных системах и формате MPEG-2

– **4:1:1** – в этом соотношении горизонтальное разрешение цветоразностных сигналов снижается до четверти от полного разрешения сигнала яркости, также полоса пропускания сужается (пропускная способность увеличивается) в два раза по сравнению с режимом без субдискретизации

– **4:2:0** – наиболее используемые в стандартных кодеках кодирования видео, таких как H26* и VC-1 и др.

– **4:1:0** - поддерживается некоторыми кодеками, но используется не слишком широко [7].

На данной картинке (Рисунок 4) используется изображение с соотношениями 4:2:0, наиболее используемый формат в кодеках, стоит отметить, что при таком подходе мы тратим всего лишь 12 бит, вместо стандартные 24 бит.

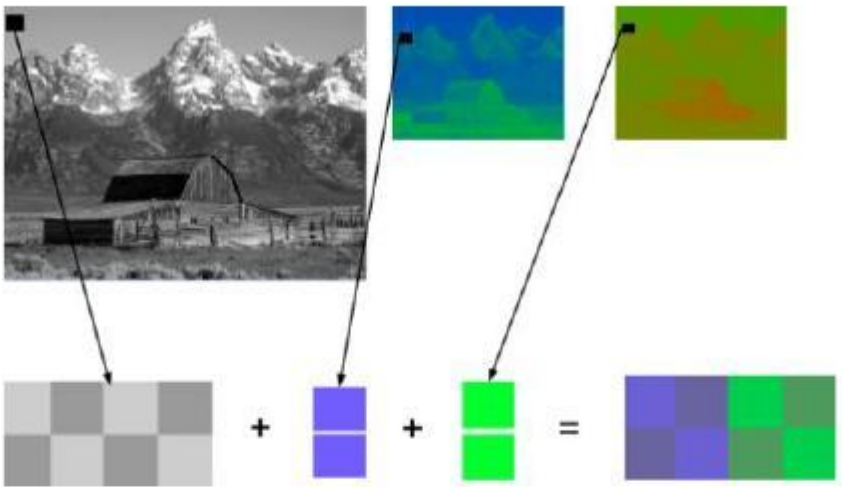

Рисунок 4 – Структура дискретизации сигнала

Вот так выглядит остальные варианты использования различных соотношений, указанные на рисунке 5.

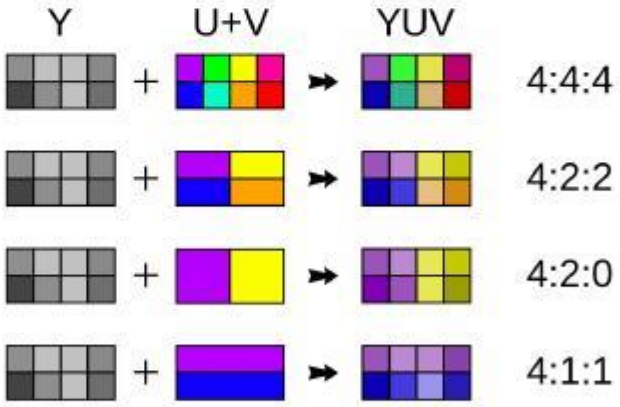

Рисунок 5 – Варианты дискретизации

Далее попробуем устранить временную избыточность. Для этого мы определим базовую терминологию. К примеру, у нас есть некий набор кадров, к примеру из 4 кадров (Рисунок 6)

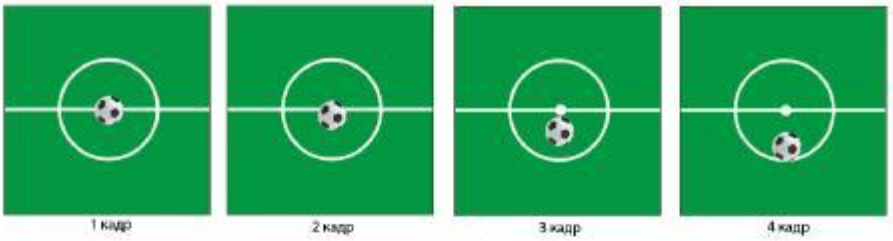

Рисунок 6 – четыре последовательных кадра

На этом рисунке можно заметить, что некоторые элементы на протяжении всего времени не меняются, как цветок на картинке, солнце или облака. Единственное, у нас падает мяч на картинке.

Intro Frame (I-кадр) – Это ключевой кадр или по-другому опорный кадр. Этот кадр является статичной фотографией. Именно на нём будут сравниваться все последующие кадры. Есть некий период во времени, в котором нужно менять опорный кадр, так как чем протяжённее мы сравниваем с одним и тем же кадром, тем больше у нас межкадровая разница, что нас может не устроить, так как возрастет пространственная избыточность, поэтому опорный кадр сменяется через какое-то N-ое количество кадров, либо до определённого порога разности.

Predicated Frame (P-кадр) – Это прогнозируемый кадр. Особенность этого кадра в том, что он сравнивает изображение с опорным кадром и находит разницу, но этот кадр полностью не будет записываться, вместо этого он будет содержать ссылку на предыдущий кадр, а также информацию об изменённой области (Рисунок 7).

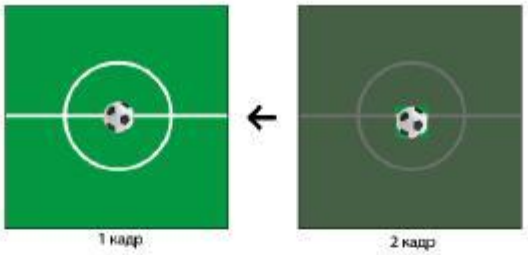

Рисунок 7 – Изменённая область

Bi-predictive Frame (B-кадр) – Это так называемый будущий кадр. Конечно, можно так же записывать только I и P кадры, но для нормального сжатия видео этого будет недостаточно, поэтому используются B кадры. Они являются самыми сжатыми по размеру и именно за счёт них сжимается львиная доля всего видеопотока. Дело в том, что он сравнивает не только предыдущий кадр, но также и следующий, так сказать будущий кадр и формируется за счёт двух соседних кадров (Рисунок 8).

Рисунок 8 – B-кадр

Классическая последовательность, обеспечивающая отличное качество с хорошим сжатием, выглядит в данном изображении (Рисунок 9):

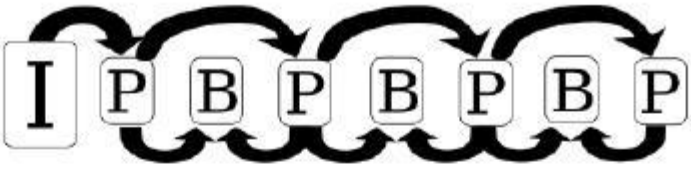

Рисунок 9 – последовательность I, P, B кадров.

Так же встречаются и другие виды последовательностей. Например, оно может выглядеть таким образом: IPBBPBBPBBPBBPBBP… Оно обозначает наиболее сильное сжатие видео из-за большего числа B-кадров [8].

Еще один способ избавления временной избыточности является **межкадровое прогнозирование**. Рассмотрим на примере разницы 1 и 2 кадра (Рисунок 10)

Рисунок 10 – покадровая разница между кадрами

В результате получаем 2 кадр и используем только разницу между этим и предыдущим кадром и кодируем лишь изменяемую область.

Такое применение имеет место быть, но сегодня используется еще более эффективный алгоритм – **Блочная компенсация движения**. Этот алгоритм разделяет текущий кадр на определённые непересекающиеся блоки, к примеру 8x8 или 16x16. Производится обход каждых блоков и каждый блок сравнивается в некотором окрестности в поиске наилучшего соответствия

между кадрами. Важно понять, что текущий кадр сравнивается с предыдущим. Таким образом мы получаем набор векторов, которые указывают движение блоков между кадрами (Рисунок 11).

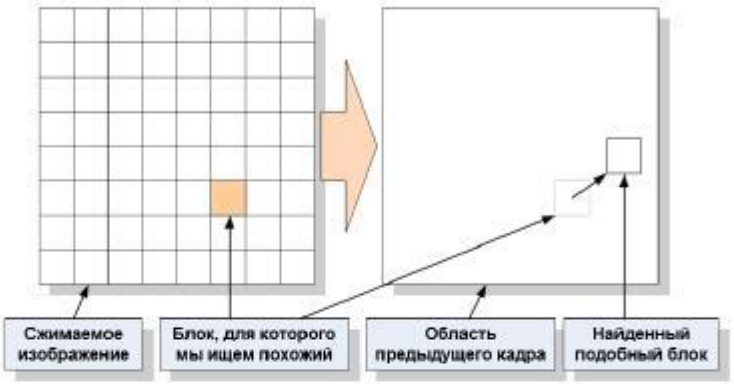

Рисунок 11 – Блочная компенсация движения

2.3.1 Обзор наиболее часто используемых кодеков

На сегодня существуют множество различных кодеков, которые используются в интернете и нужно проанализировать текущую ситуацию.

Среди них мы рассмотрим самые распространённые кодировщики, которые представляют на рынке:

– H.264 / AVC

– H.265 / HEVC

– AV1

– VP9

H264, **MPEG-4 Part 10** или **AVC** (Advanced Video Coding) — лицензируемый стандарт сжатия видео, предназначенный для достижения высокой степени сжатия видеопотока при сохранении высокого качества [9].

Это самый распространённый кодек на сегодня, он воспроизводится практически на любом устройстве, обеспечивает качественные видеопотоки и обеспечивает минимальную заботу о лицензионных платежах, что в итоге

охватывает каждого пользователя. К тому же он поддерживается практически всеми браузерами, что можно убедиться, зайти на данный сайт https://caniuse.com/. Поддерживаемые сайты приведены на рисунке 12.

Рисунок 12 – Рейтинг распространения кодека h.264

H.265 или **HEVC** (англ. High Efficiency Video Coding — высокоэффективное кодирование видеоизображений) — формат видеосжатия с применением более эффективных алгоритмов по сравнению с H.264/MPEG-4 AVC. Этот кодек был разработан в 2013 году и данный формат в два раза более эффективен, чем H.264 при кодировании. Он обеспечивает повышение эффективности сжатия на 25-50% при улучшенном или одинаковом качестве видео. Подобно AVC, он поддерживает разрешение 8K UHD, но сравнительно обеспечивает меньшие файлы, которые требуют низкой пропускной способности для потоковой передачи.

Однако он не нашел широкого распространения в интернете из-за проблем с патентами, и крупные производители попросту не желают тратиться на них, поскольку это привело бы к повышению цен для конечного потребителя. Несмотря на то, что новый стандарт гораздо эффективнее предшественника, к сожалению, он на сегодня охватывает максимум 22% браузера и мобильных устройств. Статистику можно видеть на рисунке 13.

Рисунок 13 Рейтинг распространения кодека h.265

AV1 - это относительно формат видео кодирования следующего поколения, разработанный Alliance for Open Media (AOM) вместе с поддержкой Google, Amazon, Cisco, Microsoft, Mozilla и Netflix. Это открытый видеокодек без лицензионных отчислений, который специально разработан для повышения эффективности кодирования и декодирования на 30% по сравнению с производительностью HEVC. Из-за своих малых вычислительных возможностей и возможностей быстрой аппаратной оптимизации AV1 обеспечивает высококачественное видео в реальном времени, масштабируемое на любое современное устройство с любой пропускной способностью.

Алгоритмы, используемые в AV1, намного более продвинуты, чем HEVC. Этот новый кодек предназначен для использования в WebRTC и HTML5 Web Video вместе с форматом аудиокодека Opus.

Однако у этого кодека есть существенный недостатки. Несмотря на то, что он закрепился в инфраструктуре потокового видео, он по-прежнему работает над созданием реального отпечатка и у него есть пути до широкого распространения. Проблема заключается в скорости кодирования.

В настоящее время все кодировщики, которые реализуют AV1, не могут обработать видео достаточно быстро. Для любого живого потока, после захвата видео, кодек должен быть записан через кодировщик, прежде чем его можно будет транслировать. Тест, проведенный Graphics и Video Lab Media

Group, показал, что кодирование AV1 в 2500-3000 раз медленнее, чем у конкурентов. Вторая проблема – это пока что низкая аппаратная поддержка, и по анализу рисунка 14, можно заметить, что она пока отсутствует на мобильных устройствах.

Рисунок 14 – Рейтинг распространения AV1

Аппаратная поддержка кодирования и декодирования означает, что непосредственно в процессоре реализованы интегральные схемы, специализированные для конкретных задач кодирования и декодирования.

VP9 – это бесплатная альтернатива HEVC, разработанная Google. Каждый браузер Chrome, телефоны Android и даже собственная платформа потокового видео Google YouTube поддерживают VP9 для беспрепятственной доставки видео конечным пользователям [10].

Кодек обеспечивает лучшее качество видео при той же скорости передачи данных, что и HEVC, и, таким образом, он очень эффективен для доставки видео 4K HD онлайн. Поскольку эффективность и производительность VP9 очень похожи на AV1, его часто называют AV0 (более ранняя версия AV1). VP9 предлагает лучшее качество на том же битрейте, что и H.265 / HEVC. Хотя большинство браузеров поддерживают его, устройства Apple – до недавнего времени не поддерживали этот кодек и лишь недавно добавили его частичную поддержку. График поддержки устройств показан на рисунке 15

Рисунок 15 – Рейтинг распространения кодека VP9(WebM)

Как можно заметить он поддерживается практически всеми устройствами с охватом в 94%. Однако он все еще не подходит для потокового вещания из-за медленной скорости и сложности кодирования и его можно использовать только на мощных компьютерах на сегодня, чтобы обеспечить наилучшую передачу. Поэтому он используется только в обычном видеоформате воспроизведения.

Таким образом мы рассмотрели доступные кодеки на сегодня и можно вкратце проанализировать:

H.264 или AVC - более старая версия формата кодирования видео. Однако он является самым распространённым на сегодня и именно его мы будем использовать в web-сервисе. С появлением многочисленных достижений в технологиях сжатия и новых видео технологиях, скорее всего стандарт сжатия будет медленно замещаться другим кодеком.

H.265 / HEVC, преемник AVC, существует уже почти шесть лет, но дистрибьюторы контента все еще не решаются принять его стандарты сжатия из-за дорогих и утомительных конфигураций.

Без сомнения Google VP9 – это мощный кодек, но он не получил широкого распространения из-за сложного процесса лицензирования патентов, но все еще является фаворитом в скором замещении H.264.

Учитывая извлеченные уроки VP9, Google позже решил поддержать разработку бесплатной AV1 от AOM.

2.4 Протоколы видеопередачи

Протокол – это набор правил, позволяющий осуществлять соединение и обмен данными между двумя и более включёнными в сеть компьютерами. Фактически разные протоколы зачастую описывают лишь разные стороны одного типа связи; взятые вместе, они образуют так называемый стек протоколов [11]. Видов протоколов многое количество, но нас интересует лишь те, которые умеют работать с потоковым вещанием.

Существуют два вида протоколов в видеовещании, которые используются на сегодняшний день, это:

– Потоковые протоколы;

– Сегментные протоколы;

Потоковые протоколы — это протоколы из мира p2p звонков: RTMP, webRTC, RTSP/RTP. Они отличаются тем, что пользователи договариваются о том, какой у них канал, подбирают битрейт кодека соответственно каналу. А еще у них есть дополнительные команды такого рода, как «дай мне опорный кадр». Если вы потеряли кадр, в этих протоколах вы можете заново его запросить [12].

Сегментные протоколы – протокол, который разделяет видео на определённые временные сегменты разного качества и отправляет в сеть.

Отличие сегментных протоколов в том, что никто ни с кем никак не договаривается. Они режут видео на сегменты, хранят каждый сегмент в различных качествах, и клиент сам может выбирать, какой сегмент смотреть. Каждый сегмент начинается с опорного кадра.

2.4.1 Протокол видеопередачи HLS

HLS (HTTP Live Streaming) – это протокол потоковой связи на основе HTTP, разработанный Apple для предоставления потокового вещания как по запросу, так и по запросу[13]. Это позволяет приемнику адаптировать битовую скорость мультимедиа к текущим условиям сети, чтобы поддерживать непрерывное воспроизведение с наилучшим возможным качеством.

HLS - это качественная потоковая технология, которая широко поддерживается в потоковой передаче от таких поставщиков, как Adobe, Microsoft, RealNetworks и Wowza. Он также поддерживает трансмуксинг в реальном времени на таких платформах, как Akamai.

Первоначально Apple разработала технологию потокового вещания http, чтобы поставщики контента могли отправлять живые или предварительно записанные аудио и видео на устройства Apple с помощью обычного web-сервера. На рисунке 16 показана общая структура протокола HLS

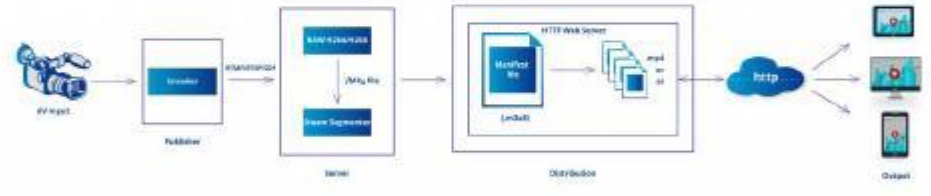

Рисунок 16 – структура протокола HLS

После того, как носитель закодирован, он сегментируется через потоковый сегментатор. Сегментер потока сегментирует мультимедийные данные, которые принимаются с фиксированным интервалом времени, генерирует сегментированный файл, а затем генерирует файл метаданных списка воспроизведения (.m3u8), через который можно получить доступ к сегментированному файлу с клиентских устройств. Прямая трансляция требует хранения данных в режиме реального времени; таким образом,

сегментирование мультимедийных данных через сегментатор не требуется. Вместо этого сегментированные потоковые данные могут быть получены на основе заранее определенной продолжительности медиа-сегмента, который затем сохраняется. Это наглядно показано на рисунке 17.

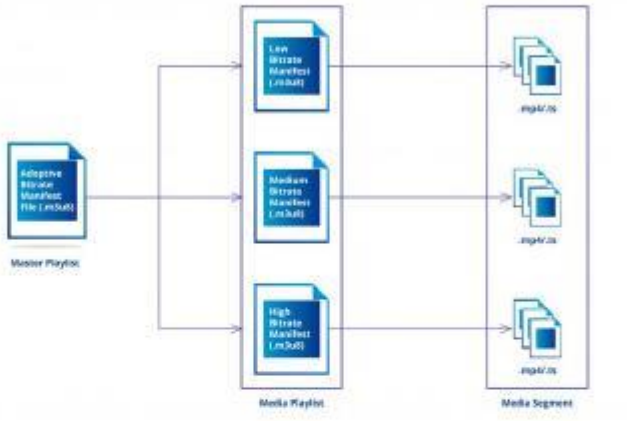

Рисунок 17 – Главный плейлист с несколькими закодированными файлами

Файл списка воспроизведения, имеющий расширение файла m3u8, содержит как минимум три типа файлов TS в медиаформате MPEG-2. Пользователи загружают расширенный файл списка воспроизведения M3U, который содержит несколько URI, соответствующих медиа-файлам. Каждый отдельный медиафайл должен быть отформатирован в формате MPEG-2 TS. Для непрерывной прямой трансляции список воспроизведения должен быть обновлен с созданием файла MPEG-2 TS вместе с созданными URI.

Передача медиаконтента по технологии HLS имеет следующие преимущества:

– Контент передается по HTTP. Это вспомогательный интернет-протокол. В этом случае такое решение работает в любом месте с доступным интернетом.

– HTTP не требует сложной настройки портов по сравнению, например, с настройкой RTSP или RTMP.

– Данные, полученные через HTTP, легко передаются через брандмауэры сторонних компаний.

– Поддерживает адаптивный битрейт для живого видео и видео по запросу (VOD). Для этого каждый зритель в любой момент может получить поток лучшего качества для своего интернет-соединения.

– Поддерживает шифрование мультимедиа и аутентификацию пользователя.

– Имеет повсеместную поддержку в различных браузерах, мобильных телефонах и операционных системах..

– Это адаптивная потоковая технология для связи с устройствами iOS и Apple. Он обеспечивает надежное, экономически эффективное средство доставки непрерывного и длинного видео через Интернет.

2.4.2 Протокол видеопередачи MPEG-DASH

MPEG-DASH – это адаптивная технология потоковой передачи битов, при которой мультимедийный контент захватывается и сохраняется на HTTP-сервере и доставляется игрокам-игрокам с использованием HTTP [14]. В этом стандарте DASH медиа-контент существует на сервере в двух частях.

– Описание мультимедийной презентации (MPD): MPD описывает манифест доступного контента, его различные альтернативы, их URL-адреса и другие характеристики.

– Сегменты. Содержит фактические мультимедийные битовые потоки в виде фрагментов, в одном или нескольких файлах.

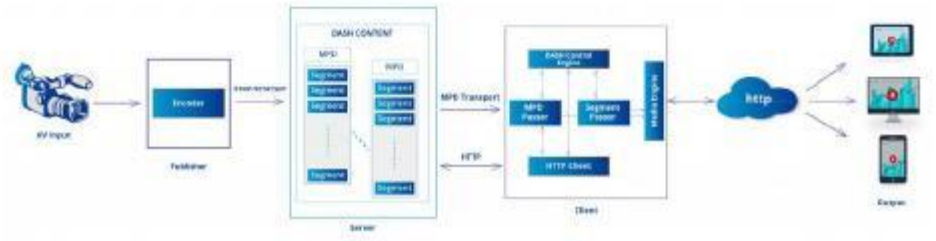

Рисунок 18 – Общая структура MPEG-DASH

Сначала HTTP-сервер предоставляет клиентским проигрывателям список доступных URL-адресов сегмента мультимедиа в файле описания мультимедийной презентации (MPD). Затем клиенты могут последовательно запрашивать сегменты мультимедиа, как требуется, для обеспечения непрерывного воспроизведения мультимедийного представления. Для воспроизведения контента DASH-клиент получает MPD, который может быть доставлен с использованием HTTP, электронной почты, флэш-накопителя, широковещательной передачи или других видов транспорта.

Анализируя MPD, клиент DASH узнает о синхронизации программы, доступности медиа-контента, типах медиа, разрешениях, минимальной и максимальной полосах пропускания и о существовании различных закодированных альтернатив мультимедийных компонентов, функций доступности и требуемого управления цифровыми правами (DRM), расположение медиа-компонентов в сети и другие характеристики контента.

На основе информации MPD клиент DASH выбирает надлежащим образом закодированный поток и отправляет HTTP-запрос для начала потоковой передачи.

Полученные сегменты мультимедиа буферизируются клиентом для обработки изменений сети. Во время воспроизведения клиент отслеживает колебания пропускной способности сети. Основываясь на этой информации, клиент решает, как адаптироваться к доступной полосе пропускания, выбирая сегменты различных альтернатив (с более низкими или более высокими битрейтами) для поддержания адекватного буфера.

MPEG-DASH позволяет использовать один и тот же поток контента на разных серверах или в сетях доставки контента (CDN). Клиент может выбрать любой из потоков, чтобы максимизировать доступную пропускную способность сети. В этом случае MPD предоставляет несколько URL-адресов для одного и того же сегмента. Продолжительность сегментов может

варьироваться как для VoD, так и для прямой трансляции. Для прямой трансляции длительность следующего сегмента также может сигнализироваться в текущем сегменте.

MPEG-DASH объединяет форматы доставки в профили DASH. Эти профили определяют функции для взаимодействия. Они идентифицируются в MPD уникальными унифицированными именами ресурсов (URN). Каждый сегмент может содержать информацию о времени UTC, поэтому клиенты могут контролировать смещение часов

DASH имеет множество преимуществ по сравнению с другими адаптивными потоковыми методами. Некоторые из них

– Спецификация DASH разработана с учетом современных отраслевых стандартов и практик.

– Он поддерживает совместимые DRM, которые позволяют устройствам разных поставщиков обмениваться данными и передавать через них.

– Предназначен для повторного использования большей части существующей инфраструктуры и инструментов.

– Это позволяет независимо использовать алгоритм управления скоростью передачи битов в клиенте, кодирование видео и аудио и метаданные.

– Он поддерживает синхронизацию сервер / клиентский компонент (например, отдельный и мультиплексированный AV)

2.4.3 Протокол видеопередачи RTMP

Real Time Messaging Protocol (сокращённо англ. RTMP) проприетарный протокол потоковой передачи данных, в основном используется для передачи потокового видео через интернет с низкой задержкой.

Это протокол на основе TCP, который поддерживает постоянные соединения и обеспечивает низкую задержку связи. Чтобы обеспечить бесперебойную доставку потоков и передачу как можно большего количества информации, он разделяет потоки на фрагменты, и их размер динамически согласовывается между клиентом и сервером. Иногда оно сохраняется без изменений; размеры фрагментов по умолчанию составляют 64 байта для аудиоданных и 128 байтов для видеоданных и большинства других типов данных.

Затем фрагменты из разных потоков могут чередоваться и мультиплексироваться по одному соединению. В случае более длинных порций данных протокол, таким образом, несет только однобайтовый заголовок на фрагмент, что приводит к очень небольшим издержкам. Однако на практике отдельные фрагменты обычно не чередуются. Вместо этого перемежение и мультиплексирование выполняются на уровне пакетов, причем пакеты RTMP по нескольким различным активным каналам чередуются таким образом, чтобы гарантировать, что каждый канал соответствует своей пропускной способности, задержке и другим требованиям качества обслуживания. Пакеты, чередующиеся таким образом, рассматриваются как неделимые и не чередуются на уровне фрагментов [15].

Так же RTMP – это метод доставки, разработанный для прямой трансляции. Это не кодек, но RTMP обычно использует кодек H.264. Проще говоря, цель живого потокового кодера состоит в том, чтобы кодировать видеопотоки в RTMP или другой протокол потоковой передачи и отправлять их на потоковый сервер для дальнейшей обработки.

Протокол RTMP больше не используется для доставки потоков зрителям, поскольку технология flash от adobe уже является устаревшим и не поддерживается браузерами. Чаще всего используется современный протокол, такой как HLS или MPEG-DASH и является предпочтительным, но RTMP по-прежнему является стандартом для доставки ваших потоков на вашу онлайн-

видео платформу, эти два инструмента работают вместе, чтобы сделать возможным передачу потокового вещания [16].

Таким образом мы рассмотрели все основные протоколы, которые используются для массового вещания. Мы выяснили, для того чтобы транслировать наш видеопоток, необходима связка RTMP и HLS или RTMP и MPEG-DASH. RTMP необходим, для того, чтобы доставлять транслируемую информацию в реальном времени с аппаратного устройства на сервер, там он обрабатывается, а HLS или MPEG-DASH создаёт файлы манифестов и доставляет по HTTP видеопоток.

3 Разработка и проектирование WEB-сервиса потокового вещания

Web-сервис представляет собой стриминговую платформу для массового вещания. Что позволит любому человеку вещать трансляцию через интернет, а другие пользователи смогут наблюдать за этим процессом и комментировать происходящее в чат. Сам web-сервис будет создан на основе уже имеющихся инструментах, модулях, библиотеках и фреймворках, упрощающие и ускоряющие разработку. Поскольку создание сервиса с нуля может оказаться очень затратным решением и крайне неэффективным.

3.1 Технологические решения web-сервиса

3.1.1 Выбор языка программирования

Основной язык разработки web-сервиса является только JavaScript. Поскольку язык поддерживает событийно-ассинхронную модель ввода/вывода. Эту модель как раз поддерживает и NODE.js, который позволяет исполнять код на серверной части приложения.

JavaScript на сервере может быть показаться очень странным и необычным решением, поскольку этот язык де-факто является клиентским языком разработки. В данный момент Node.js ничуть не уступает в

производительности, чем PHP. Это связано с наследием Google, который сделал огромный вклад в производительность JavaScript движка — V8. В PHP и большинство других серверных языков используют очевидную модель блокировки. Когда вы делаете запрос на извлечение информации из базы данных, запрос выполнит и завершит процесс, прежде чем перейдет к следующему оператору. В Node.js все иначе. В Node.js не нужно ждать. Вместо этого можно создать функцию обратного вызова, которая, прослушивая процесс, выполняется после того, как действие завершится.

Возможность применять один язык и на сервере, и в клиенте позволяет упростить и ускорить разработку полноценных web-приложений. Так же, помимо этого, есть огромное число сторонних модулей и различных решений для разработки.

Однако нельзя и сказать о недостатках, и в каком случае проект не годится с использованием NODE.JS. Так как JavaScript – однопоточный и ассинхронный язык, то есть использует неблокирующие операции ввода/вывода, то его использование будет уместно в том случае, если обычно его используют там, где надо держать много одновременно открытых соединений, каждое из которых не требует много процессорного времени, но может тормозить из-за ввода/вывода.

Таким образом, именно поэтому уместно использовать NODE.JS в качестве нашего web-приложения.

Однако в остальных случаях, где нужны сложные научные вычисления или не используются много соединений одновременно, то разумно использовать другие решения для разработки.

3.1.2 Основные инструменты и расширения разработки сервиса

Для реализации web-сервиса были использованы следующие основные технологии:

– Node.js – основная среда разработки приложения. Так же его можно использовать в качестве web-сервера при установке модулей. Главная причина выбора технологии – поддержка ассинхронно-событийных механизмов.

– Nmp – менеджер пакетов, который входит в состав Node.js. Он позволит нам устанавливать модули разработки к нашему приложению.

Основные модули, которые используются в приложении при помощи npm:

– Express.js – это минималистичный фреймворк для разработки серверной части web-приложения.

– Handlebars.js – шаблонизатор, используется для отображения html кода, путём генерации шаблонов на сервере.

– Socket.io – JavaScript-библиотека для web-приложений и обмена данными в реальном времени. Состоит из двух частей: клиентской, которая запускается в браузере и серверной для node.js. Используется для чата.

– Mongoose – это библиотека JavaScript, часто используемая в приложении Node.js для работы с базой данных MongoDB.

– Node-media-server – основной rtmp сервер для обработки видеопотока.

3.1.3 Обзор используемых внешних программ

Используются две основных программы для реализации web-сервиса:

– Visual Studio Code – Основной рабочий инструмент, редактор исходного кода.

– OBS – свободная программа с открытым исходным кодом для записи видео и потокового вещания, разрабатываемая проектом OBS и сообществом независимых разработчиков.

– FFmpeg – набор свободных библиотек с открытым исходным кодом, которые позволяют записывать, конвертировать и передавать цифровые аудио- и видеозаписи в различных форматах

3.1.4 Выбор базы данных

Основная база данных для данного сервиса является MongoDB. Выбор этой базы данных является не совсем стандартным и скорее является экспериментальным подходом к разработке самого сервиса, но это не означает, что это будет верным решением при дальнейшей разработке и масштабирования самого приложения.

Основная причина, почему выбрана эта база данных:

– Отсутствие сложных связей.

– Использование сущностей, имеющие лишь характер записи событий, то есть документирование или логирование и не требует сложной агрегации. Примером такой сущности из web-сервиса является сохранение сообщений в чате.

– Так же одним из преимуществ является гибкий JSON-формат документов. Так как web-сервис полностью состоит из JavaScript, то такой подход к сохранению данных более чем уместен.

– Высокая производительность и гибкость запросов. В NoSql базах данных не принято использовать сложные агрегации, хоть такая возможность существует. То есть, если нам нужно сделать что-то вроде JOIN, мы можем сходить выбрать данные, потом сходить выбрать данные по ссылкам и затем их обработать на стороне приложения.

3.1.5 Серверная архитектура web-сервиса

Основная структура web-сервиса будет состоять из двух серверов, которые будут прослушиваться на разных портах одного компьютера и одной

базы данных MongoDB. Первый сервер отвечает за взаимодействие с клиентами, то есть HTTP сервер. Второй сервер отвечает за прием и раздачу видеопотоков – RTMP сервер. Общая схема выглядит на рисунке 19. В силу финансовых ограничений, для таких случаев используют отдельные сервера, которые будут выполнять только свою задачу.

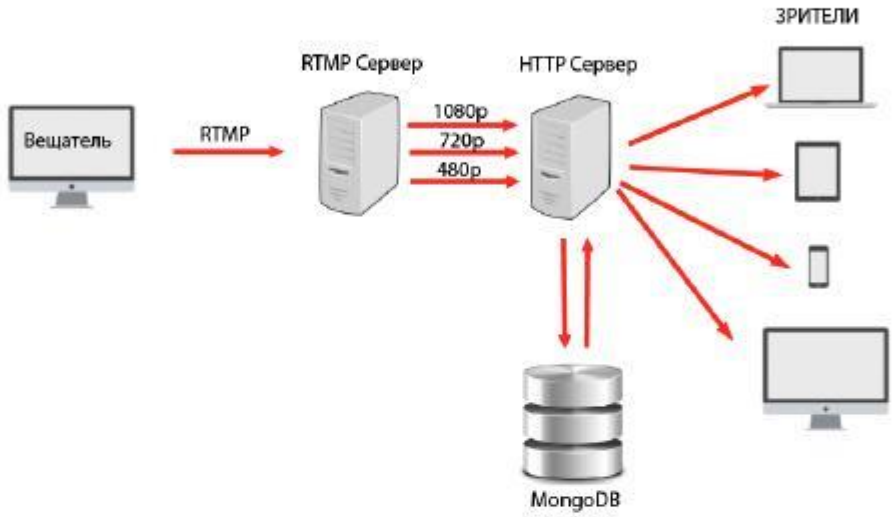

Рисунок 19 – Схема архитектуры серверной части

3.2 Разработка web-сервиса.

3.2.1 Структура разрабатываемого приложения

При организации web-сервиса необходимо разработать, а точнее использовать некий паттерн разработки для упрощения навигации и чистоты стиля кода. На данный момент уже есть давно устоявший паттерн под названием **MVC**. Он разделяет наш код на три компонента: модель, представление, контроллер. Воспользуемся данным паттерном и организуем изначальную структуру, изображенная на рисунке 20

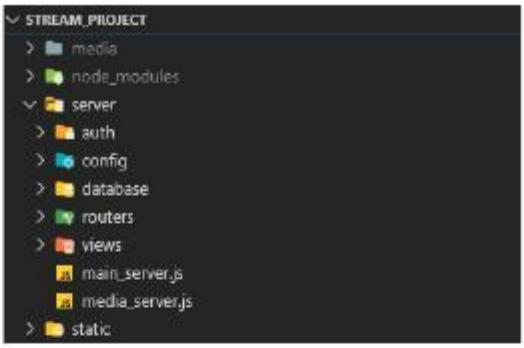

Рисунок 20 – Структура папок

Разберёмся за что будут содержать наши папки и файлы:

– *media* – папка с динамическим содержимым. Сюда включается всё то, что создаётся непосредственно во время работы приложения, а именно сюда будут генерироваться наши манифест-файлы для воспроизведения трансляций;

– *node_modules* – временная папка для разработки самого приложения. Здесь содержатся используемые модули для разработки;

– *server* – основная папка серверной части приложения. Далее идут подпункты, которые содержатся в этой папке.

- *auth* – папка, которая отвечает за регистрацию/авторизацию пользователей;
- *config* – папка, отвечающая за конфигурацию сервера;
- *database* – папка, содержащая основные модели базы данных в приложении;
- *routers* – папка, отвечающая за навигацию в приложении или по другому – роутинг;
- *views* – папка, содержащая шаблоны для отображения web-сервиса;
- *main_server.js* – главный файл запуска приложения, запускает http сервер;

– *media_server.js* – тоже главный файл, запускает rtmp сервер;

– *static* – папка со статическим содержимым (css, js, т.д).

3.2.2 Структура клиентской части

Web-сервис служит для отображения трансляций зрителям, поэтому нужно сосредоточить внимание зрителя на основных функциях web-сервиса. С главной страницы нужно обеспечить доступ к трансляциям в данный момент. Для действующих пользователей должна быть возможность быстрой авторизации, а после авторизации – доступ к своим данным с любой страницы сайта. Основная клиентская структура описана на рисунке 21.

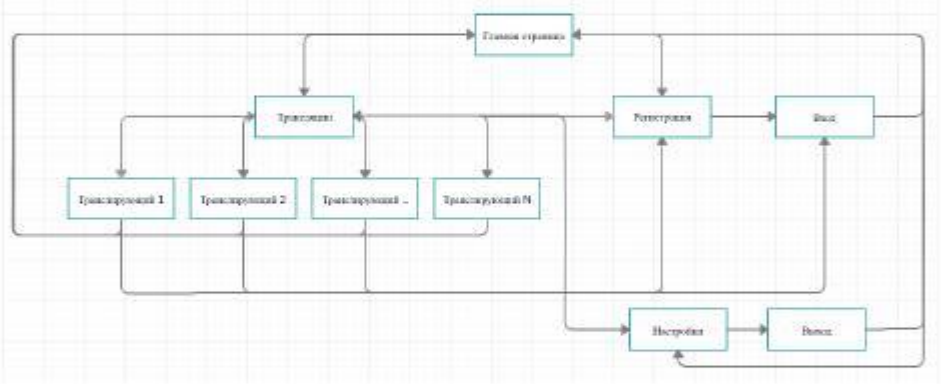

Рисунок 21 – Структура клиентской части

3.2.3 Реализация модели базы данных

Для реализации базы данных используется mongoose – это библиотека ORM (object-relational mapping), которая связывает базу данных с языками языков программирования, по-другому можно назвать драйвером. Заранее условимся, что в NoSql базах, вместо привычного использования таблиц в реляционных базах, используются документ как структура. Базы данных NoSQL подходят к моделированию с другой точки зрения в целом.

Сам web-сервис состоит из двух документов: User и Streams. В привычном понимании здесь документ представляется как отдельная сущность. Сама схема документов представлена на рисунке 22.

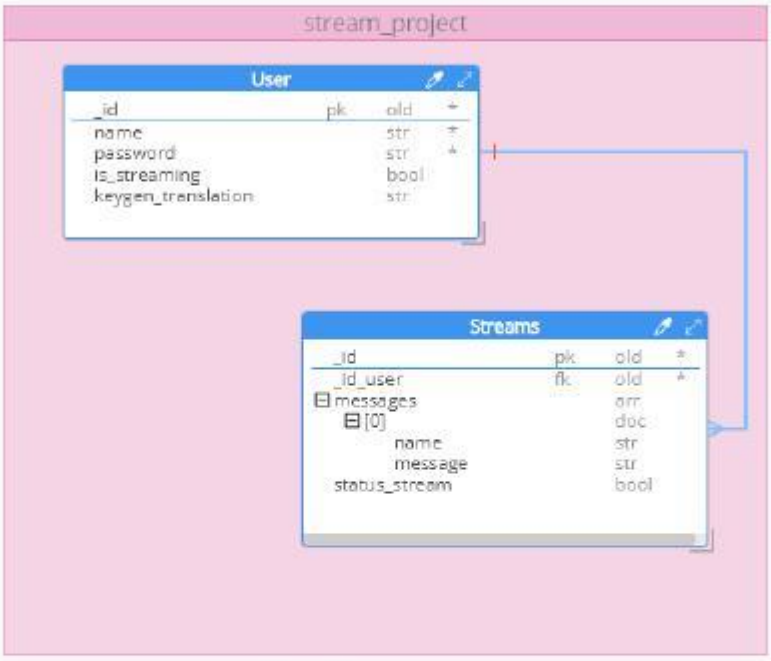

Рисунок 22 – Структура базы данных

Рассмотрим используемые поля в этих таблицах:

User – информация о пользователях:

– *_id* – уникальный идентификатор пользователя

– *name* – никнейм/имя пользователя

– *password* – пароль пользователя

– *keygen_translation* – уникальный ключ для возможности транслирования

– *is_streaming* – статус транслирования пользователя

Streams – документ, содержащий информацию о трансляции пользователей

- *_id* – уникальный идентификатор трансляции
- _id_user – id пользователя, из документа User, который транслирует
- *messages* – массив сообщений в данной трансляции
  - *name* – пользователь сообщения
  - *message* - сообщение
- *status_stream* – статус трансляции.

Чтобы создать схему, создадим три файла в папке *database* и реализуем по вышеописанной схеме. Файл *UserSchema.js*:

```
let Schema = require('mongoose').Schema;
let md5 = require('js-md5');
let shortId = require('shortid');

let User = new Schema({
    name: String,
    password: String,
    keygen_translation: String,
    isStreaming: Boolean
});

User.methods.cryptPassword = (password) => {
    return md5(password);
}

User.methods.validPassword = (chekedPassword) => {
    return md5(chekedPassword);
}

User.methods.generateKey = () => {
    return shortId.generate();
}
module.exports = User;
```

Помимо этого, в этом классе, мы так же добавляем дополнительные методы *cryptPassword()* и *validPassword()* которые отвечают за кодирование и проверки пароля по стандарту хешировании **md5**. И еще третий метод *generateKey()*, который генерирует короткий ключ для того, чтобы обеспечить доступ к трансляции для пользователя. Так же стоит заметить, что поле *_id*, которое отвечает за уникальный идентификатор, оно создаётся автоматически и в коде оно используется неявно.

Файл *StreamsSchema.js*:

```
let mongoose = require('mongoose');
let Schema = require('mongoose').Schema;


module.exports = new Schema({
    _id_user: mongoose.Types.ObjectId,
    messages: Array,
    status_stream: Boolean
});
```

В этом файле описывается схема Streams.

Файл *Shema.js*:

```
let mongoose = require('mongoose');

module.exports = {
    User: mongoose.model('User', require('./UserSchema')),
    Streams: mongoose.model('Streams', require('./StreamsShema')),
}
```

В этом файле импортируются наши модели и создаются в нашей базе данных mongoDB. Далее впоследствии этот файл тоже будет импортируемым, чтобы мы могли обращаться к нашей базе данных через описанные уже наши модели.

3.2.4 Вёрстка web-сервиса

Поскольку JavaScript не может самостоятельно генерировать шаблоны статически на стороне сервера, для этого используются специальные шаблонизаторы, которые считывают файл и заменяют содержимое. В итоге отдавая готовый html файл для клиента. В данном случае мы используем шаблонизатор **handlebars**. Все файлы с шаблонами будут храниться в директории *views*. Главная страница будет иметь только содержательную информацию о web-сервисе, а также иметь навигацию. Сама главная страница изображена на рисунке 23.

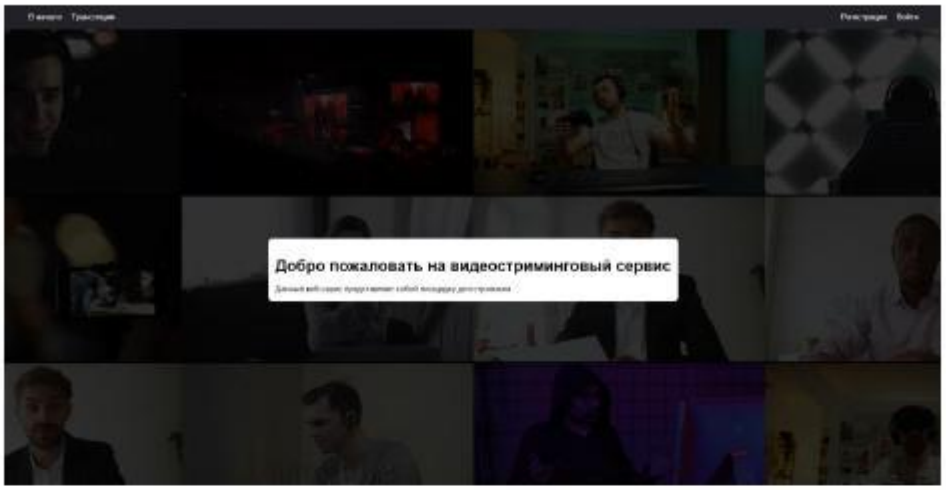

Рисунок 23 – Главная страница web-сервиса

Поскольку позже будет описана авторизация и регистрация пользователей, соответственно для них так же нужно создать отдельные шаблоны. Разработанный шаблон для авторизации, представлен на рисунке 24.

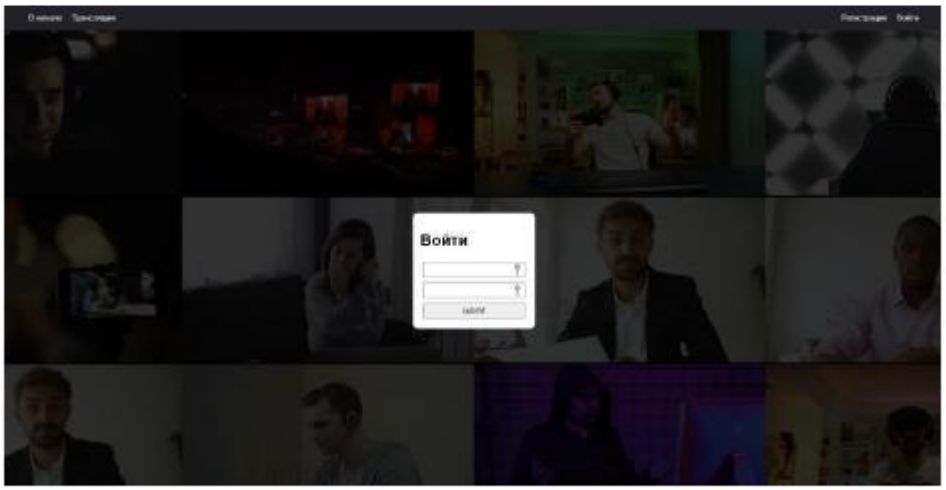

Рисунок 24 – Страница авторизации

Аналогично представлен шаблон регистрации.

В случае, если пользователь авторизован, то он видит шапку в правом верхнем углу экрана, представленную на рисунке 25.

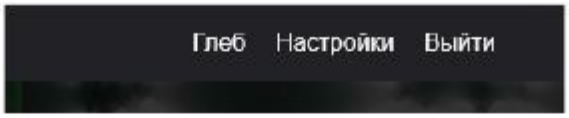

Рисунок 25 – Навигационная панель при авторизации

Далее, на вкладке «Трансляции» находятся доступные трансляции пользователей в данный момент, то есть те, кто вещает онлайн. Данный шаблон будет состоит из множества элементов, которые располагаются в ряд. Данная вёрстка представлена на рисунке 26. Каждый зритель, даже без авторизации может зайти на пользовательскую трансляцию.

Рисунок 26 – Страница с трансляциями

Так же есть блок с настройками пользователя, сама настройка предполагает только наличие генерируемого ключа для трансляции. Сам шаблон реализован на рисунке 27. Пользователь так же может сгенерировать новый ключ во избежание утечки.

Рисунок 27 – Страница с настройками

Последний шаблон, это соответственно сама трансляция. Именно так выглядит этот шаблон, указанный на рисунке 28. Он состоит из видеоплеера в центральной части и блока с чатом, чтобы зрители могли обсуждать трансляцию или получать обратную связь.

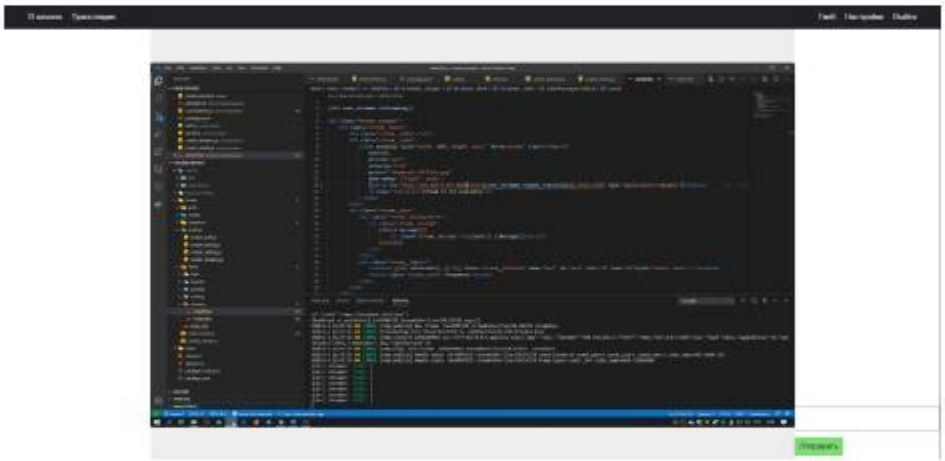

Рисунок 28 – Страница трансляции пользователя

В целом можно подчеркнуть, что дизайн довольно прагматичный и минималистичный и отражает лишь только основные принципы и функции самого web-сервиса. Все шаблоны находятся в необязательном рассмотрении ПРИЛОЖЕНИЯ В.

3.2.5 Реализация регистрации и авторизации web-сервиса

Для того, чтобы была возможность транслировать свой канал, нужна система регистрации пользователей и выдавать секретный ключ. Для этого используется библиотека passport.js. Библиотека является промежуточным ПО для аутентификации Node.js. Чрезвычайно гибкий и модульный, Passport может быть незаметно вставлен в любое web-приложение на основе Express. Полный набор стратегий поддерживает аутентификацию с использованием имени пользователя и пароля, Facebook, Twitter и т. Д.

Поскольку мы уже создали вышеописанные модели для пользователя, то нам осталось только подключить библиотеку и описать функции регистрации и авторизации. Создадим файл auth.js в папке auth, которую мы создали ранее и напишем код:

```
let passport = require('passport');
let LocalStrategy = require('passport-local').Strategy;
let User = require('../database/Schema').User;
```

В первых 2 строчках подключаем модули. Модуль passport-local означает то, что мы будем создавать свою систему авторизации самостоятельно, из наших полей user и password. Однако существуют и другие решения авторизации. Например, если бы нам понадобилось авторизация из социальных сетей, то можно подключить модуль passport-vkontakte или passport-facebook. Таких модулей очень много и, если нам придется расширить сервис, то без проблем можно использовать их. Далее мы регистрируем наши стратегии авторизации и регистрации, соответственно LocalLogin и LocalRegistrer. Эти функции ассинхронные и обращаются к базе данных.

```
passport.use('LocalRegister', new LocalStrategy(async function(username, password
, done){
    try{
        let err, user = await User.findOne({name: username});
        if(err) { console.log(err); return done(err); }
        if(user){
            return done(null, false);
        }else{
            let user = new User();
```

```
            user.name = username;
            user.password = user.cryptPassword(password);
            user.keygen_translation = user.generateKey();

            let err, usr = await user.save();
            if(err) throw err;
            return done(null, usr);
        }
    }catch(err){ console.log(err) }
}))

passport.use('LocalLogin', new LocalStrategy({
        usernameField: 'username',
        passwordField: 'password'
    }, async function(username, password, done){
        let err, user = await User.findOne({name: username});

        if(err) { console.log(err); done(err); }
        if(!user) { return done(null, false); }
        if(user){
            if(user.password === user.validPassword(password)){
                return done(null, user);
            }else{
                return done(null, false);
            }
        }
    }
))
```

Далее следующие 2 функции:

– serializeUser() которая отвечает за то, что пользователь находится в одной сессии текущего браузера.

– deserializeUser(), которая отвечает за выход пользователя из аккаунта и убирает его из текущей сессии.

```
passport.serializeUser(function(user, done) {
    return done(null, user.id);
  });

passport.deserializeUser(async function(id, done) {
    let err, user = await User.findById(id);
    return done(err, user);
});

let isAuth = function(req, res, next){
    if(req.isAuthenticated()) { return next(); }
    return res.redirect('/');
```

```
}
```

Последней строкой мы делаем модуль импортируемым для использования.

```
module.exports = {
    passport: passport,
    isAuth: isAuth
}
```

В целом модуль авторизации уже готов, осталось только определить маршруты, для которых эти стратегии регистрации будут работать. Соответственно в папке routers, созданную ранее, и который представляет собой контроллер, определим маршруты. Создадим файл router_auth.js и определим следующие маршруты. Мы импортируем созданный нами модуль авторизации и дополнительную функцию isAuth(), созданная ранее, которая проверяет пользователя, что он авторизован.

```
let router = require('express').Router();
let passport = require('../auth/auth').passport;
let isAuth = require('../auth/auth').isAuth;
```

Далее мы создаем маршрут /register с доступами GET и POST. При простом GET запросе мы возвращаем просто созданный ранее шаблон. При POST, первым аргументом мы определяем обработчик LocalRegister. Вторым – дополнительные параметры, при успешной регистрации, successRedirect перенаправляет нас на страницу с авторизации пользователя, в противном случае failureRedirect оставляет нас на той же странице с ошибкой.

```
router.route('/register')
    .get((req, res) => {
        res.render('auth/register');
    })
    .post(passport.authenticate('LocalRegister', {
        successRedirect: '/auth/login',
        failureRedirect: '/auth/register'
    }))
```

Маршрут /login представляет собой почти тот же паттерн, только вместо регистрации – соответсвенно обработчик авторизации LocalLogin.

```
router.route('/login')
    .get((req, res) => {
        res.render('auth/login');
    })
    .post(passport.authenticate('LocalLogin', {failureRedirect: '/auth/login'}),
function(req, res){
        res.redirect('/');
    })
```

Маршрут /logout отвечает за выход пользователя из аккаунта.

```
router.get('/logout', isAuth, (req, res) => {
    req.logOut();
    res.redirect('/');
})
module.exports = router
```

3.2.6 Реализация HTTP сервера

Главный файл main_server.js содержит в себе абсолютно все подключаемые файлы, содержит настройки конфигурации самого сервера, подключения базы данных и является главной входной точкой запуска приложения. Этот файл является важным (подробнее см. в Приложении А)

3.2.7 Реализация RTMP сервера

Прежде чем настраивать rtmp сервер, нам нужно установить дополнительное ПО – ffmpeg. Это очень гибкий кодек для работы с видео. Он поддерживает множество стандартов сжатия как видео, так и аудио, имеет информативный анализатор и настройки. Его возможный недостаток – это то, что он имеет только консольный интерфейс, вместо графического и из-за этого у него может быть большой порог вхождения.

Для реализации rtmp сервера нам понадобится дополнительный модуль под названием node-media-server. Это open-source модуль, в котором реализуется rtmp протокол и принимает входящие, как правило сжатые пакеты видеопотока в кодировке h.264 и далее с помощью ffmpeg программа создаёт

файл(-ы) манифес(-тов) .h3u8(HLS) или .mpd(MPEG-DASH) и их содержимое .ts и .m4s соответственно. После этого содержимое может отправляться в публичный доступ для клиентов.

Перед созданием сервера нам нужно изначальные настройки для работы с сервером. Создадим файл server.js в папке config и внесём следующие настройки:

```
const servers = {
    rtmp_server: {
        rtmp: {
            port: 1935,
            chunk_size: 60000,
            ping: 60,
            ping_timeout: 30
        },
        http: {
            port: 8888,
            mediaroot: './media',
            allow_origin: '*'
        },
        trans: {
            ffmpeg: 'E:/SSDSofts/ffmpeg/bin/ffmpeg.exe',
            tasks: [{
              app: 'live',
              hls: true,
              hlsFlags: '[hls_time=2:hls_list_size=2:hls_flags=delete_segments]',
              dash: false,
            }]
        }
    },
};

module.exports = servers;
```

В словаре rtmp_server будут содержаться наши основные настройки rtmp сервера. Далее в самом словаре присутствуют еще 3 ключа с названиями rtmp, http, trans. Разберём по порядку.

Ключ rtmp содержит следующие ключ-значение:

– port – это стандартный значение rtmp протокола, по которому будет прослушиваться порт.

– chunk_size - максимальный размер порции для мультиплексирования потока

– ping – пинг соотвественно;

– ping_timeout – максимальная задержка по времени, если пинг превысит стандартные значения.

Ключ http отвечает за дополнительный сервер и служит для отдачи наших манифест файлов. Содержит следующие ключ-значение

– port – прослушиваемы порт http дополнительного сервера.

– mediaroot – директория, куда отправляются генерируемые файлы.

– allow_origin – это Access-Control-Allow-Origin указывает, какие домены могут обращаться к ресурсам сайта. В данном случае выставлены по умолчанию все с целью разработки. Однако при развертывании указывается только текущий адрес web-сервиса.

Ключ trans содержит настройки для программы ffmpeg. Использует следующие ключ-значение:

– ffmpeg – основной путь к запуску исполняемого файла программы ffmpeg

Ключ tasks содержит основные настройки для ffmpeg:

– app – название задачи, а так же URL-путь от корневого домена, по которому будут доступны файлы манифесты.

– hls – Содержит булевое значение и включает режим протокола HLS.

– hlsFlags – определяет 3 параметра: первый установливает длину целевого сегмента в секундах. Второй устанавливает максимальное количество записей в плейлисте. Третий отвечает за удаление сегментов после завершении трансялции.

– dash – Так же можно включить режим и MPEG-DASH, но не будем использовать.

Создадим файл media_server.js в папке server и импортируем используемые модули:

```
let NodeMediaServer = require('node-media-server');
let fs = require('fs');
let path = require('path');
let spawn = require('child_process').spawn;
```

```
let ffmpeg = require('./config/server').rtmp_server.trans.ffmpeg;
let config = require('./config/server').rtmp_server;
let User = require('./database/Schema').User;
let Streams = require('./database/Schema').Streams;
```

Перечислим необходимые подключаемые модули:

– node-media-server – главный модуль, который создаёт rtmp сервер

– fs – модуль для работы с файловой системой.

– path – библиотека для удобства работы с местоположением файлов.

– spawn – подключаем функцию из модуля child_process. Этот модуль отвечает за создание дочерних процессов в системе. Функция spawn как раз создаёт в системе этот процесс.

Остальные настройки уже импортируются из известных модулей, описанных ранее.

Сам сервис node-media-server состоит из обработчиков обратного вызова – слушателей, которые нужно нам определить. Всего существует 9 обработчиков:

– preConnect – обработчик, который вызывается перед соединением сервера.

– postConnect – обработчик, который вызывается после соединения с сервером.

– doneConnect – обработчик, который вызывается после завершения соединения.

– prePublich – обработчик, который вызывается перед публикацией манифест файлов

– postPublish – обработчик, который вызывается в момент генерации манифест файлов.

– donePublish – обработчик, который вызывается после завершения публикации файлов.

– prePlay – обработчик, который вызывается до начала подготовки к трансяции.

– postPlay – обработчик, который вызывается к готовности трансялции.

– donePlay – обработчик, который вызывается после завершении трансляции.

Для текущих задач нам достаточно всего 2 обработчика: prePublish и doneConnect. Перед публикацией трансляции и после завершения трансляции.

Первый обработчик будет описываться таким кодом:

```
nms.on('prePublish', async function(id, StreamPath, args) {
    console.log('[NodeEvent on prePublish]', `id=${id} StreamPath=${StreamPath} a
rgs=${JSON.stringify(args)}`);
    let key = StreamPath.split('/');
    let id_key = key[key.length - 1];

    let err, user = await User.findOne({"keygen_translation": id_key});

    if(err){
        console.log(err);
    }else if(user){

        let err, usr = await User.findOne({"keygen_translation": id_key});
        let stream = new Streams({
            "_id_user": usr._id,
            "messages": [],
            "status_stream": true
        })
        stream.save()
        err, update = await User.updateOne({"keygen_translation": id_key}, {$set:
 {"isStreaming": true}});


        thumbnails_interval = setInterval(function(){
            fs.access(path.join(path.dirname(__dirname), '/media/thumbnails/'+use
r.name), (err) => {
                if(err){
                    fs.mkdir(path.join(path.dirname(__dirname), '/media/thumbnail
s/'+user.name), (err) => {
                        create_thumbnails(id_key, user.name);
                    })
                }
                create_thumbnails(id_key, user.name);
            })
        }, 10000);
    }else{
        console.log('user not find');
        let session = nms.getSession(id);
        session.reject();
    }
```

```
});
```

Эта функция принимает в себя 3 аргумента, идентификатор стрима, путь публикации и дополнительные аргументы. В этой функции мы получаем путь нашего секретного ключа для того, чтобы найти пользователя и потом создать новый элемент стрима в базу данных. Так же в этой функции генерируются миниатюры каждые 10 секунд для заставки в разделе сайта «Трансляции», чтобы пользователь видел предварительную картинку транслируемого потока.

Вторая функция отвечает за завершение трансляции и описывается следующим кодом:

```
nms.on('donePublish', async function(id, StreamPath, args){
    let key = StreamPath.split('/');
    let id_key = key[key.length - 1];
    let err, user, update;
    await User.updateOne({"keygen_translation": id_key}, {$set: {"isStreaming": f
alse}});

    err, user = await User.findOne({"keygen_translation": id_key});
    err, update = await Streams.updateOne({"_id_user": user._id, "status_stream":
 true}, {"status_stream": false});
    console.log(err, update);

    clearInterval(thumbnails_interval);
})
```

Здесь мы ищем пользовательскую трансляцию по данным пользователя, а точнее его секретного ключа и тем самым обновляем статус на завершение трансляции и прекращаем задачу на генерацию миниатюр.

Сама функция генерации миниатюр выглядит следующим образом:

```
function create_thumbnails(id_key, name){
    spawn(ffmpeg,
        ['-y',
        '-i', 'http://127.0.0.1:8888/live/'+id_key+'/index.m3u8',
        '-ss', '00:00:01',
        '-vframes', '1',
        '-vf', 'scale=-2:300',
        path.join(path.dirname(__dirname), '/media/thumbnails/'+name+'/live.png')
,
    ])
}
```

Мы создаём дочерний процесс с помощью ffmpeg, который будет обрезать фрагмент кадра и уменьшая его в размере.

В целом основной RTMP сервер настроен и завершён. Осталось настроить клиентскую часть для того, была возможность воспроизводить трансляцию.

3.2.8 Реализация видеоплеера с использованием библиотеки video.js

Поскольку стандартный html5 с тегом video не поддерживает воспроизведение протоколов HLS или MPEG-DASH в реальном времени, то существуют различные решения, которые решают эти проблемы на языке JavaScript. Одним из таких решений будет плеер от video.js. Для того, чтобы имелась возможность воспроизводить наше содержимое, добавим в шаблон detail.hbs следующий код:

```
<div class="stream__video">
       <video autoplay style="width: 100%; height: auto;" id="my-
player" class="video-js"
                controls
                preload="auto"
                autoplay="true"
                poster="/thumbnails/Offline.png"
                data-setup='{"fluid": true}'>
                <source src="http://localhost:8888/live/{{user_streamer.keygen_tr
anslation}}/index.m3u8" type="application/x-mpegURL"></source>
           <p class="vjs-no-js">Stream is not available</p>
       </video>
</div>
 <script src="/scripts/video.js/dist/video.js"></script>
```

В последней строчке мы подключаем плеер и в теге <source> указываем путь к нашему манифест файлу. Так же нужно указать MIME тип – «application/x-mpegURL» для воспроизведения. В целом плеер подключен и готов обрабатывать нашу трансляцию.

3.3 Демонстрация работы web-сервиса

После основных описанных шагов, стоит продемонстрировать, как работает сам сервис.

Первым шагом будет регистрация на web-сервисе, поэтому необходимо создать аккаунт на сайте. После регистрации мы должны авторизоваться под своими данными и зайти на вкладку «настройки». На этой странице (Рисунок 29) мы увидим наш секретный ключ, чтобы получить возможность транслировать на web-сервисе.

Рисунок 29 – Секретный ключ для трансляции

После того, как мы получили наш секретный ключ, необходимо скачать ПО, которое бы умело записывать наш видеопоток со входных устройств. В данном случае используется широко используемая программа OBS, которая представлена на рисунке 30.

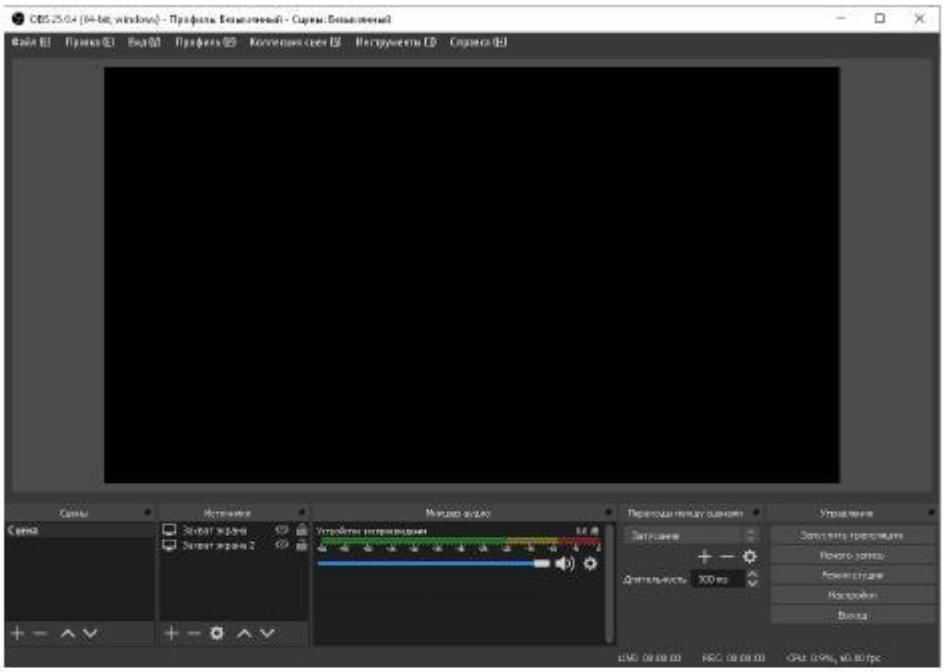

Рисунок 30 – Программа OBS для захвата области транслирования

Далее нам нужно настроить, куда транслировать наши данные, в данном случае видеопоток будет транслироваться с экрана (Рыисунок 31)

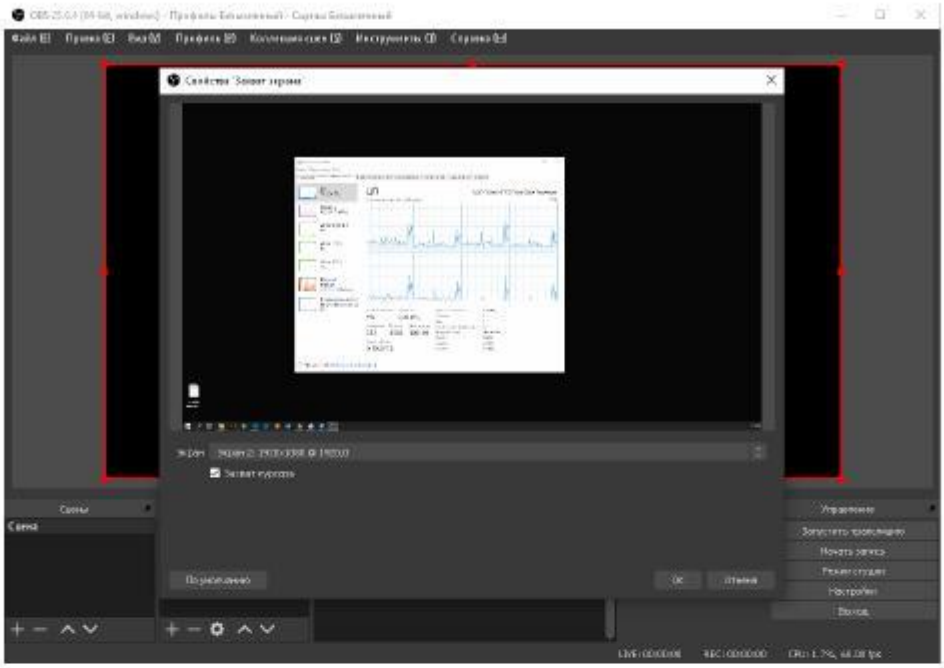

Рисунок 31 – Захват рабочего стола

После этого необходимо зайти во вкладку «настройки» и выбрать пункт «вывод». Там присутствует окно «потоковое вещание» (Рисунок 32)

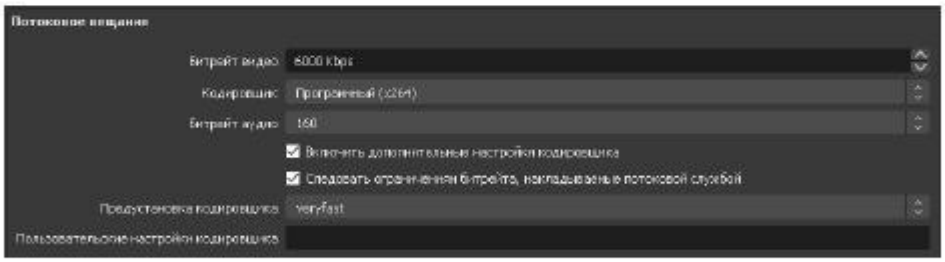

Рисунок 32 – Настройка кодировщика

Значение битрейта нужно выбирать в соответствии производительности компьютера, то есть насколько мощное железо находится у вас. И зависит от видеокарты, либо от вашего процессора. Значение самого битрейта определяет

качество вашего потока и определённых значений точных нет, существует примерно условные значения, приведённые в данной таблице:

Таблица 1 – Различные настройки битрейта.

Разрешение	Низкое качество	Среднее качество	Высокое качество
360p и 25 FPS	500 Kbps	800 Kbps	1200 Kbps
480p и 25 FPS	750 Kbps	1200 Kbps	1600 Kbps
480p и 30 FPS	1000 Kbps	1400 Kbps	1800 Kbps
720p и 30 FPS	2000 Kbps	2750 Kbps	3500 Kbps
1080p и 30 FPS	3200 Kbps	6000 Kbps	9000 Kbps
1080p и 60 FPS	5500 Kbps	8000 Kbps	14000 Kbps

Как видно, параметр битрейта сугубо субъективный. Мы выставили битрейт на 6000 Kbps. Этого вполне достаточно для комфортного просмотра большинства контента, с разумным использованием ресурсов компьютера.

Далее нам нужно выбрать кодировщик. В данном случае здесь присутствуют 2 пункта: Программный (x264) и Аппаратный (NVENC). Первый использует ресурсы CPU, то есть процессора. Второй – возможности видеокарты. В данном случае стоит 8 ядерный процессор от AMD FX8350 и ресурсов процессора вполне достаточно для кодирования.

Следующим шагом необходимо зайти во вкладку «настройки» и выбрать пункт «вещание». Здесь мы видим 3 поля: сервис, сервер и ключ потока. В списке сервис присутствуют множество площадок трансляций, таких как YouTube, Twitch, Periscope и многие другие. Нас интересует именно пункт «настраиваемый». Далее нам необходимо указать сервер куда будет передаваться наш видеопоток. В данном случае мы указываем свой сервер с протоколом rtmp. И последнее поле необходимо указать наш ключ трансляции в целях безопасности, чтобы никто не мог взломать наш видеопоток и не мог

вести трансляцию из-под нашего аккаунта. Пример заполненных данных указаны на рисунке 33.

Рисунок 33 – Настройка данных для трансляции

В целом основные шаги уже закончены и осталось только запустить трансляцию. На главном экране нужно нажать «запустить трансляцию». После этого на сайте по верхней ссылке «трансляции» будет доступен наш поток (Рисунок 34) и каждый может присоединиться и смотреть.

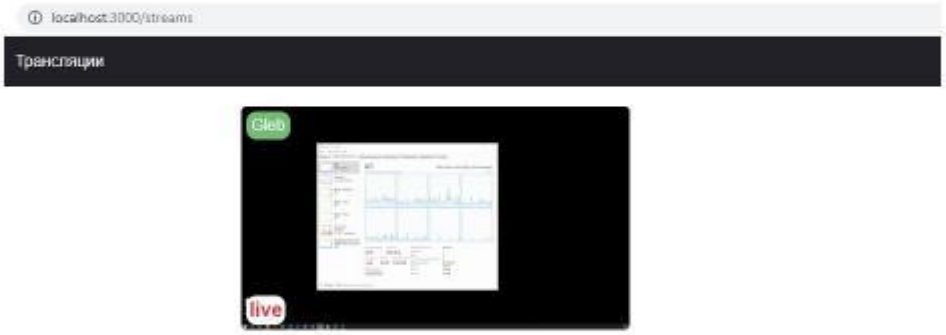

Рисунок 34 – Итоговая трансляция на web-сервис

ЗАКЛЮЧЕНИЕ

С каждым годом увеличивается приток пользователей в интернет, происходит развитие самого интернета и различных сфер жизни. Всё больше людей используют эти возможности и множество профессий, не только в сфере программирования, получают возможность работать из дома. Повышается спрос на медийную сферу. Люди транслируют из дома и проводят различные конференции, спортивные и киберспортивные игры. Студенты проводят трансляции с преподавателями. Важность потокового вещания на сегодня имеет высокий приоритет.

В этой работе было рассмотрено множество материалов по организации web-сервиса потокового вещания. Сначала мы рассмотрели актуальность самого вещания в интернете и сферу применения.

Далее были изучены принципы кодирования и выбор основного кодека кодирования. Рассмотрели основные используемые на сегодня протоколы доставки видеоизображения.

Целью данной бакалаврской работы являлась разработка web-сервиса потокового вещания с помощью NODE.JS. Этот web-сервис рассчитан на публичное транслирование.

В ходе разработки самого web-сервиса была изучена предметная область, проанализированы требования к сервису. Были выбраны основные инструменты разработки и осуществили иной подход к базе данных. Была организована схема передачи видеоконтента. Так же разработана регистрация и авторизация, и интерфейс клиентской части.

Сам web-сервис не предполагает серьёзного внедрения и разработан исключительно с целью изучить новейшие технологии и возможности его использования. Реализованный проект с легкостью может быть расширен новыми возможностями и функциями.

В ходе разработки мною был изучен большой объем документации как клиентских, так и серверных технологий. Получены навыки в разработке

клиент-серверных приложений и верстке web-страниц с использованием современных технологий.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Потоковое вещание (Live streaming). Общая информация [Электронный ресурс]. – [Б. м. : б. и.]. – Режим доступа: https://itmultimedia.ru/potokovoe-veshhanie-live-streaming-obshhaya-informaciya/ - Загл. с экрана.

2 Гарматин, А.А. Интернет вещание в системе и СМИ: Особенности и принципы функционирования [Электронный ресурс] / А.А. Гарматин. – Электрон. текстовые дан. – Вестник ВГУ, 2004. – Режим доступа: http://www.vestnik.vsu.ru/pdf/phylolog/2004/02/garmatin.pdf

3 Технология потокового видео и сферы ее применения [Электронный ресурс]. – [Б. м. : б. и.]. – Режим доступа: https://www.redmax.tv/blog/potokovoe-video/. - Загл. с экрана.

4 Дмитрий Ватолин. Алгоритмы сжатия видео [Электронный ресурс] / Дмитрий Ватолин // Национальный Открытый Университет «ИНТУИТ» – [Б. м. : б. и.]. – Режим доступа: https://www.intuit.ru/studies/courses/1069/206/lecture/5338?page=1. - Загл. с экрана.

5 Дмитрий Ватолин. Введение в сжатие видео [Электронный ресурс] / Дмитрий Ватолин // Московский Государственный Университет CSMSU Graphics&Media Lab – [Б. м. : б. и.], 2005. – Режим доступа: https://www.graphicon.ru/oldgr/courses/mdc/lectures/2005/compr_video_v.2.4.pdf . – Загл. с экрана.

6 Цветовая субдискретизация [Электронный ресурс]. – [Б. м. : б. и.]. – Режим доступа: https://ru.wikipedia.org/wiki/Цветовая_субдискретизация - Загл. с экрана.

7 Как работает видеокодек. Часть 1. Основы [Электронный ресурс] – [Б. м. : б. и.], 2019. – Режим доступа: https://habr.com/ru/company/edison/blog/481418/. - Загл. с экрана

8 Сжатие видео. I кадры, P кадры, B кадры [Электронный ресурс] – [Б. м. : б. и.], 2017. – Режим доступа: https://www.sdelayvideo.ru/page/szhatie-video-i-kadry-r-kadry-b-kadry/. – Загл. с экрана.

9 Сетевые протоколы [Электронный ресурс] – [Б. м. : б. и.]. – Режим доступа: http://bourabai.kz/dbt/protocols.htm/ – Загл. с экрана.

10 Traci Ruether. Video Codecs and Encoding: Everything You Should Know [Электронный ресурс] / Traci Ruether // WOWZA media systems. – [Б. м. : б. и.], 2019. – Режим доступа: https://www.wowza.com/blog/video-codecs-encoding/. – Загл. с экрана.

11 AV1-HEVC-VP9: Which is the best quality video codec for online streaming [Электронный ресурс] – [Б. м. : б. и.], 2019. – Режим доступа: https://www.muvi.com/blogs/best-video-codec-for-streaming.html/. – Загл. с экрана.

12 Пишем свой протокол поверх UDP [Электронный ресурс]. – [Б. м. : б. и.], 2018. – Режим доступа: https://habr.com/ru/company/oleg-bunin/blog/413479/. - Загл. с экрана.

13 What Is HLS (HTTP Live Streaming)? How HLS Works? [Электронный ресурс] – [Б. м. : б. и.]. – Режим доступа: https://www.synopi.com/hls-http-live-streaming/. - Загл. с экрана.

14 What Is MPEG DASH (Dynamic Adaptive Streaming Over HTTP)? [Электронный ресурс] – [Б. м. : б. и.]. – Режим доступа: https://www.synopi.com/mpeg-dash/. – Загл. с экрана.

15 Real-Time Messaging Protocol [Электронный ресурс] – [Б. м. : б. и.], – Режим доступа: https://en.wikipedia.org/wiki/Real-Time_Messaging_Protocol/. – Загл. с экрана.

16 Max Wilbert. How to Setup an RTMP Encoder for Live Video Streaming [Электронный ресурс] / Max Wilbert // Dacast - Video Hosting & Live Streaming Solutions. – [Б. м. : б. и.], 2020. – Режим доступа: https://www.dacast.com/blog/rtmp-encoder-live-video-streaming/. – Загл. с экрана.

ПРИЛОЖЕНИЕ А (обязательное)

Основной входной файл main_server.js

```
let express = require('express');
let hbs = require('./config/handlebars').setting;
let path = require('path');
let http = require('http');
let nms = require('./media_server');
let passport = require('./auth/auth').passport;
let DB = require('./config/database');
let bodyParser = require('body-parser');
let session = require('./config/session');
let config = require('./config/server');


let app = express();
let server = http.createServer(app);
let io = require('./socket_server')(server);

app.engine('hbs', hbs.engine);
app.set('view engine', 'hbs');
app.set('views', path.join(__dirname, './views'));

app.use('/scripts', express.static('node_modules'))
app.use('/media', express.static('media'))
app.use('/thumbnails', express.static('media/thumbnails'))
app.use(express.static('static'));

//промежуточная обработка сокета req и res
io.use(function(socket, next){
    session(socket.request, socket.request.res, next)
})
app.use(session);
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

let sessionPassportInit = passport.initialize();
let sessionPassport = passport.session();
io.use(function(socket, next){
    sessionPassportInit(socket.request, socket.request.res, next)
})
io.use(function(socket, next){
    sessionPassport(socket.request, socket.request.res, next);
});
app.use(sessionPassportInit);
app.use(sessionPassport);


app.use('/auth', require('./routers/router_auth'));
app.use('/streams', require('./routers/router_streams'));
app.use('/setting', require('./routers/router_setting'));
```

```
app.use('/media', require('./routers/router_media'));

app.get('/', async (req ,res) => {
    let context = { user: req.user }
    res.render('index', context);
})

server.listen(config.port, () => {
    console.log(`server lister on port: http://${config.ip}:${config.port}`);
    nms.run();
})
```

ПРИЛОЖЕНИЕ Б (обязательное)

Конфигурация сервера и различных преднастроек.

Листинг 1. (Подключение к базе данных) database.js

```
let mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1/stream', {useNewUrlParser: true, useUnified
Topology: true});
mongoose.connection.on('error', console.error.bind(console, 'DATABASE CONNECTION
ERROR'));
mongoose.connection.once('open', function() { console.log("DATABASE CONNECTED");
});

module.exports = mongoose;
```

Листинг 2. (Настройка шаблонизатора) handlebars.js

```
let hbs = require('express-handlebars');

let setting = hbs.create({
    layoutsDir: 'server/views/layouts',
    defaultLayout: 'default',
    extname: '.hbs'
});

setting._renderTemplate = function(template, context, options){
    options.allowProtoMethodsByDefault = true;
    options.allowProtoPropertiesByDefault = true;

    return template(context, options);
}

module.exports = {
    hbs: hbs,
    setting: setting
}
```

Листинг 3. (Настройки сервера) server.js

```
const servers = {
    port: 3000,
    rtmp_server: {
        rtmp: {
            port: 1935,
            chunk_size: 60000,
            gop_cache: true,
            ping: 60,
            ping_timeout: 30
```

```
        },
        http: {
            port: 8888,
            mediaroot: './media',
            allow_origin: '*'
        },
        trans: {
            ffmpeg: 'E:/SSDSofts/ffmpeg/bin/ffmpeg.exe',
            tasks: [{
                    app: 'live',
                    hls: true,
                    hlsFlags: '[hls_time=2:hls_list_size=2:hls_flags=delete_segme
nts]',
                    dash: false,
                    dashFlags: '[f=dash:window_size=3:extra_window_size=5]'
            }]
        }
    },
};

module.exports = servers;
```

Листинг 4. (Настройка сессии) session.js

```
let session = require('express-session');
let MongoStore = require('connect-mongo')(session);

module.exports = session({
    secret: 'secret',
    resave: false,
    saveUninitialized: true,
    store: new MongoStore({
        url: 'mongodb://localhost/sessions',
        autoRemove: 'enabled'
    })
})
```

ПРИЛОЖЕНИЕ В (необязательное)

Листинг 1 (базовый шаблон) default.hbs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link rel="stylesheet" href="https://necolas.github.io/normalize.css/8.0.1/norm
alize.css">
  <link rel="stylesheet" href="/scripts/video.js/dist/video-js.min.css">
  <link rel="stylesheet" href="/css/style.css">
  <title>Сервис потокового вещания</title>
</head>
<body>
    <div id="app">
      <nav class="nav-bar">
        <ul class="nav-bar__items">
          <li class="nav-bar__item"><a href="/">В начало</a></li>
          <li class="nav-bar__item"><a href="/streams">Трансляции</a></li>
        </ul>
        <ul class="nav-bar__items">
           {{#if user}}
          <li class="nav-bar__item"><a>{{user.name}}</a></li>
          <li class="nav-
bar__item"><a href="/setting/{{user.name}}">Настройки</a></li>
          <li class="nav-bar__item"><a href="/auth/logout">Выйти</a></li>
           {{else}}
          <li class="nav-bar__item"><a href="/auth/register">Регистрация</a></li>
          <li class="nav-bar__item"><a href="/auth/login">Войти</a></li>
          {{/if}}
        </ul>
      </nav>
      <section class="content">{{{body}}}</section>
    </div>
</body>
</html>
```

Листинг 2. (Шаблон для главной страницы) preview.hbs

```
<div class="preview__wrapper">
    <div class="preview__videos">
        <div><video autoplay loop><source src="/preview-
video/1.webm" type="video/webm"></video></div>
        <div><video autoplay loop><source src="/preview-
video/2.webm" type="video/webm"></video></div>
        <div><video autoplay loop><source src="/preview-
video/3.webm" type="video/webm"></video></div>
```

продолжение ПРИЛОЖЕНИЯ В

```
        <div><video autoplay loop><source src="/preview-
video/4.webm" type="video/webm"></video></div>
        <div><video autoplay loop><source src="/preview-
video/15.webm" type="video/webm"></video></div>
        <div><video autoplay loop><source src="/preview-
video/17.webm" type="video/webm"></video></div>
        <div><video autoplay loop><source src="/preview-
video/9.webm" type="video/webm"></video></div>
        <div><video autoplay loop><source src="/preview-
video/8.webm" type="video/webm"></video></div>
        <div><video autoplay loop><source src="/preview-
video/9.webm" type="video/webm"></video></div>
        <div><video autoplay loop><source src="/preview-
video/10.webm" type="video/webm"></video></div>
        <div><video autoplay loop><source src="/preview-
video/11.webm" type="video/webm"></video></div>
        <div><video autoplay loop><source src="/preview-
video/12.webm" type="video/webm"></video></div>
    </div>
     <div class="preview__background"></div>
</div>
```

Листинг 3. (Шаблон с настройками пользователя) setting.hbs

```
<div class="settings__wrapper">
    <div class="settings__block">
        <h1>Настройки: </h1>
        <form method="POST" class="settings__form">
            <h3>Секретный ключ для трансляции: </h3>
            <input type="text" name="key_stream" id="key_stream" value="{{key_gen
}}">
            <input type="submit" value="Сгенерировать ключ">
        </form>
    </div>
</div>
```

Листинг 4. (Шаблон с выборкой трансляции) index.hbs

```
<div class="streams__wrapper">

    <div class="streams__items">
        {{#if streaming}}

            {{#each users}}
                <div class="streams__item">
                    {{#if isStreaming}}
                        <p class="streams__name">{{name}}</p>
                        <a class="streams__link" href="/streams/{{name}}">
```

```
                        <img class="streams__img" src="/thumbnails/{{name}}/l
ive.png" alt="" width="400">
                    </a>
                    <p class="streams__live">live</p>
                {{/if}}
            </div>
        {{/each}}
    {{else}}
        <h1>В данный момент трансляций нет</h1>
    {{/if}}
</div>
</div>
```

Листинг 5. (Шаблон пользовательской трансляции) detail.hbs

```
{{#if user_streamer.isStreaming}}

<div class="stream__wrapper">
    <div class="stream__block">
        <div class="stream__lobby"></div>
        <div class="stream__video">
            <video autoplay style="width: 100%; height: auto;" id="my-
player" class="video-js"
                controls
                preload="auto"
                autoplay="true"
                poster="/thumbnails/Offline.png"
                data-setup='{"fluid": true}'>
                <source src="http://192.168.0.247:8888/live/{{user_streamer.keyge
n_translation}}/index.m3u8" type="application/x-mpegURL"></source>
                <p class="vjs-no-js">Stream is not available</p>
            </video>
        </div>
        <div class="stream__chat">
            <div class="stream__dialog-block">
                <ul class="stream__dialog">
                    {{#each messages}}
                        <li class="stream__message"><p>{{user}}: {{message}}</p><
/li>
                    {{/each}}
                </ul>
            </div>
            <div class="stream__inputs">
                <textarea {{#if isUserAuth}} {{/if}} class="stream__textarea" nam
e="text" id="text" cols="30" rows="10"style="resize: none;"></textarea>
                <button class="stream__enter">Отправить</button>
            </div>
```

продолжение ПРИЛОЖЕНИЯ В

```
        </div>
    </div>
</div>

    <script src="/scripts/video.js/dist/video.js"></script>
    <script>videojs.options.flash.swf = "video-js.swf";</script>
    <script src="/socket.io/socket.io.js"></script>
    <script src="/js/streams_detail_socket.js"></script>

{{/if}}
```